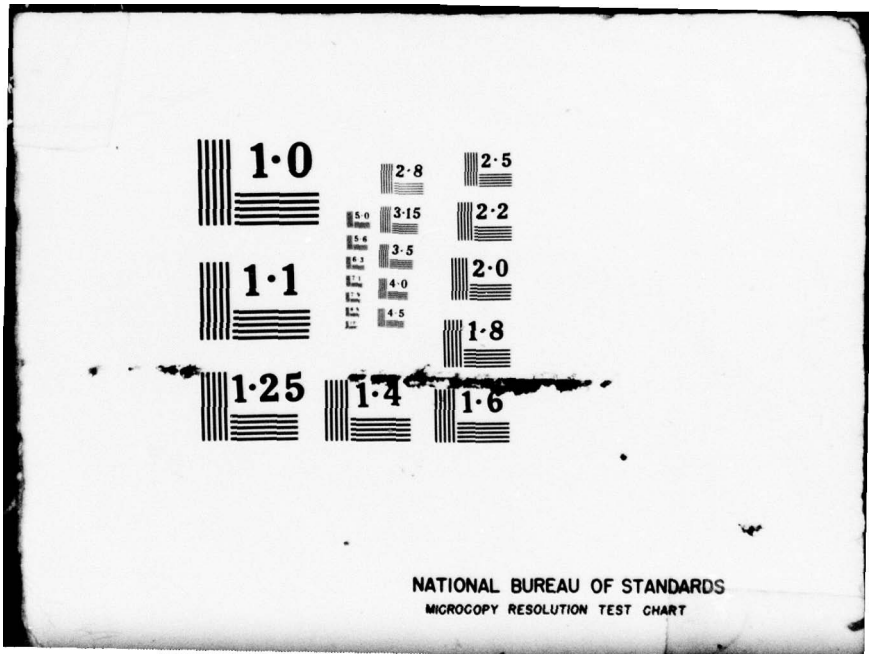
AD-A068 204 DAVID W TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CE--ETC F/G 9/2
GIRS (GRAPH INFORMATION RETRIEVAL SYSTEM) USERS MANUAL.(U)
APR 79 I S ZARITSKY

UNCLASSIFIED DTNSRDC-79/036 NL

1 OF 3
ADA
068204


1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·5
2·8
3·15
3·5
4·0
4·5
5·0
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

LEVEL IV

DAVID W. TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CENTER

Bethesda, Maryland 20084

GIRS
(GRAPH INFORMATION RETRIEVAL SYSTEM)
USERS MANUAL

by

Irving S. Zaritsky

DDC
RECEIVED
MAY 4 1979
C

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

COMPUTATION, MATHEMATICS, AND LOGISTICS DEPARTMENT
RESEARCH AND DEVELOPMENT REPORT

April 1979

DTNSRDC-79/036

AD A068204

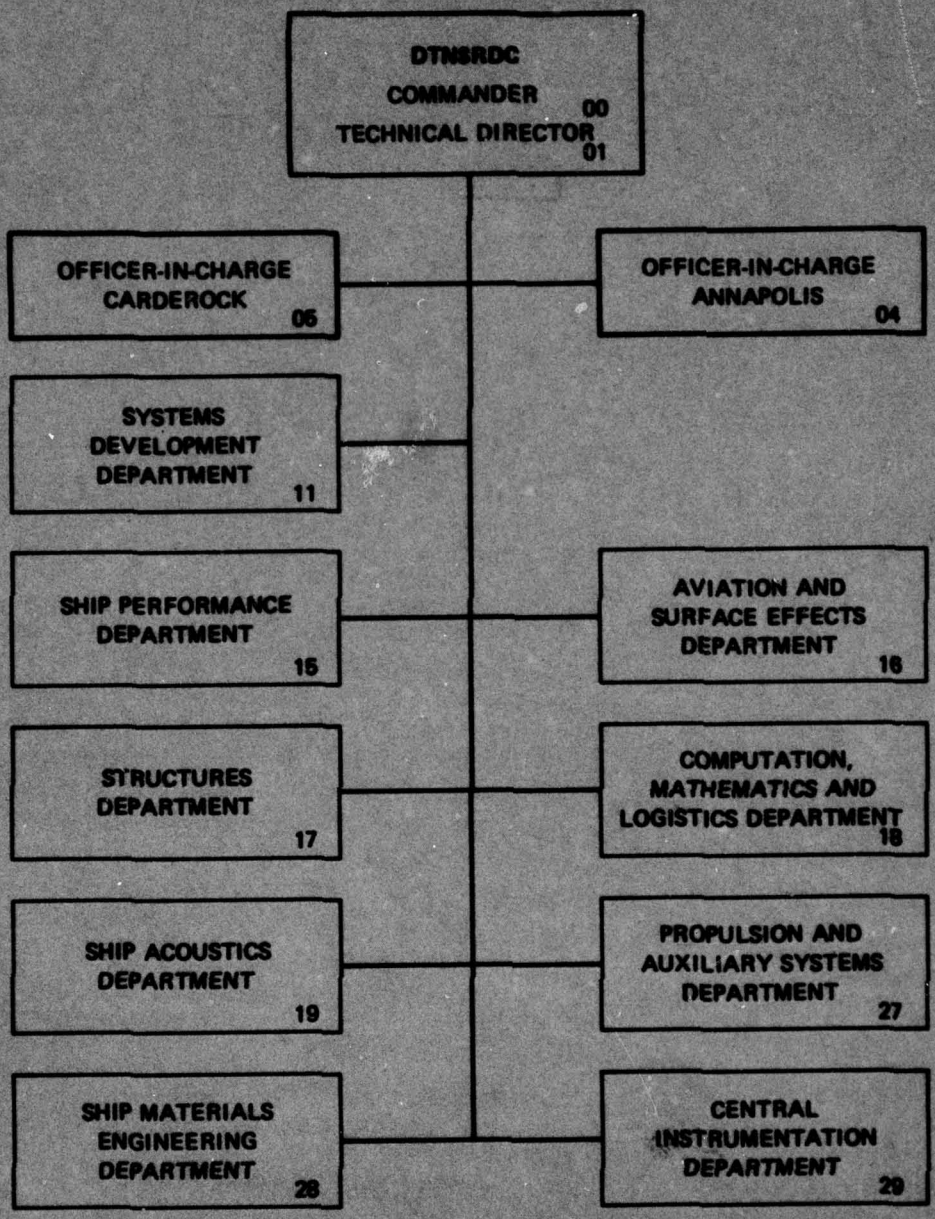
DDC FILE COPY

DTNSRDC-79/036

GIRS (GRAPH INI

79 05 03 005

MAJOR DTNSRDC ORGANIZATIONAL COMPONENTS



NDW-DTNSRDC 3960/43b (Rev. 3-78)
GPO 928 740

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: DTNSRDC-79/036 (14)	2. GOVT ACCESSION NO. ✓
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE *(and Subtitle)* (6): GIRS (GRAPH INFORMATION RETRIEVAL SYSTEM) USERS MANUAL.	5. TYPE OF REPORT & PERIOD COVERED
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10): Irving S. Zaritsky	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: David W. Taylor Naval Ship Research and Development Center, Bethesda, Maryland 20084	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: (See reverse side)
11. CONTROLLING OFFICE NAME AND ADDRESS (11)	12. REPORT DATE: Apr[redacted]79
	13. NUMBER OF PAGES: 194
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* (12) 192p.	15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

(9) Research and development rept.,

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Associative Memory	Graph
Data Base	Hashed Addressing
Data Definition Language	Information Retrieval System
Data Management	Data Base Management System

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The Graph Information Retrieval System (GIRS) provides a convenient and efficient technique for the insertion, retrieval, modification, and deletion of data in a data base. This technique is based on a scheme of representing the various data items as nodes and establishing the relationships between the nodes by linking them together into node-link-node triples which are assembled into a graph that can be stored on disk. GIRS

(Continued on reverse side)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

387 682

79 05 03 005

(Block 10)

Program Element 6276N
Project F53532
Task Area ZF53532001
Work Unit 1809-009

(Block 20 continued)

is made up of 15 FORTRAN subroutines. It has been implemented on the CDC 6700, the PDP 11/45, and the UNIVAC 1108 computing systems, and can easily be adapted to other machines having FORTRAN IV compilers. The implementation and use of GIRS are described.

TABLE OF CONTENTS

	Page
LIST OF TABLES	v
ABSTRACT	1
INTRODUCTION	1
COMPUTER REPRESENTATION OF THE GRAPH STRUCTURE	6
THE FOUR FIELDS OF WRKSPC	6
INITIALIZATION OF THE GIRS BUFFER	6
REPRESENTATION OF NODES AND LINKS	8
COMPUTATION OF A BUFFER ADDRESS FOR A FUNCTION	8
THE CONFLICT LIST	8
THE FLAG FIELD	10
THE SINGLE-VALUED FUNCTION	11
THE MULTIVALUED FUNCTION	11
HOLLERITH DATA	14
INTEGER DATA	14
SAMPLE GIRS STRUCTURE	15
USE OF GIRS SUBROUTINES	18
INITIALIZATION	18
Operating Sequences	18
Subroutine LVSETP	24
Subroutine LVGRN	26
RETRIEVAL OF VALUES	29
Subroutine LVFIND	29
Subroutine LVFNV	32
RETRIEVAL OF MVL INDEX OF GIVEN VALUE OF A FUNCTION (INCLUSION)	38
Subroutine LVINCL	38
INSERTION	40
Subroutine LVNSRT	40
DELETION	46
Subroutine LVDLET	46

Page

DISK STORAGE AND RETRIEVAL OF A GRAPH..... 49
Subroutine LVDUMP..... 50
Subroutine LVFECH..... 51

COMPRESSION/EXPANSION OF GIRS BUFFER SPACE..... 52
Subroutine LVCMPN..... 53

CONVERSION OF GIRS BUFFER FROM A PACKED
TO AN UNPACKED VERSION, AND VICE-VERSA..... 57
Subroutine LVUNPK..... 58
Subroutine LVPACK..... 59

INTERNAL ROUTINES..... 60
Subroutine LVUPDT..... 60
Function LVRTSH..... 61
Function LVLFSH..... 62

DECK SETUPS AND COMMAND SEQUENCES..... 63

GENERAL DISCUSSION..... 63

INDIRECT USE OF GIRS SUBROUTINES VIA GIRL..... 63

DIRECT USE OF GIRS SUBROUTINES..... 68

PROPOSED EXTENSIONS TO GIRS..... 69

ACKNOWLEDGMENTS..... 70

APPENDIX A - SAMPLE PROGRAMS..... 71

APPENDIX B - EXECUTION TIMES OF THE BASIC
GIRS OPERATIONS..... 131

APPENDIX C - GIRS SUBROUTINES USED ONLY
WITH THE GIRL PREPROCESSOR..... 133

APPENDIX D - ALGORITHM FOR GENERATING THE
SEQUENCE OF RANDOM NUMBERS..... 135

APPENDIX E - MEMORY REQUIREMENTS..... 137

APPENDIX F - VARIABLES IN LABELED COMMON..... 141

APPENDIX G - SUBROUTINE LISTINGS..... 143

REFERENCES..... 187

LIST OF TABLES

	Page
1 - The Flag Field	10
2 - Maximum Number of Hollerith Characters Allowed Per Triple	14
3 - Maximum Size of Integer Values That May be Stored Per Triple	14
4 - Memory Requirements for GIRS and for the GIRL Preprocessor	138
5 - Lengths of GIRS Subroutines	139

ABSTRACT

The Graph Information Retrieval System (GIRS) provides a convenient and efficient technique for the insertion, retrieval, modification, and deletion of data in a data base. This technique is based on a scheme of representing the various data items as nodes and establishing the relationships between the nodes by linking them together into node-link-node triples which are assembled into a graph that can be stored on disk. GIRS is made up of 15 FORTRAN subroutines. It has been implemented on the CDC 6700, the PDP 11/45, and the UNIVAC 1108 systems and can easily be adapted to other machines having FORTRAN IV compilers. The implementation and use of GIRS are described.

INTRODUCTION

Efficient information access for a computer user must rely upon some effective method of representing data relationships within the computer. Of the various schemes developed for this purpose, a particularly useful one is the Graph Information Retrieval System (GIRS).[1*] GIRS provides several notable capabilities:

- Placement of a value at any location within a multivalued list (nondestructive insertion)
- Substitution of a value within any location within a multivalued list (destructive insertion)
- Deletion of a value within any location within a multivalued list (indexed** deletion)
- Retrieval of a value from any location within a multivalued list
- Retrieval of the index associated with a particular sink node value (submitted by the user) within a multivalued list
- Reduction of retrieval time needed for long lists through the use of internally saved indices
- Adjustment of the GIRS buffer size resulting in a more efficient use of computer space
- Convenient disk storage and retrieval of entire graphs
- Ease of adaptation to other machines having FORTRAN compilers. It has already been implemented on the CDC 6700, the PDP 11/45, and the UNIVAC 1108 computer systems.

---

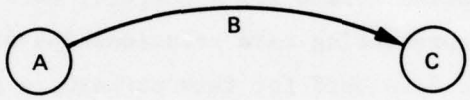
*A complete listing of references is given on page 187.

**The index of a value is the position it occupies within the list.

All of these features are described in further detail in the section "Use of GIRS Subroutines."

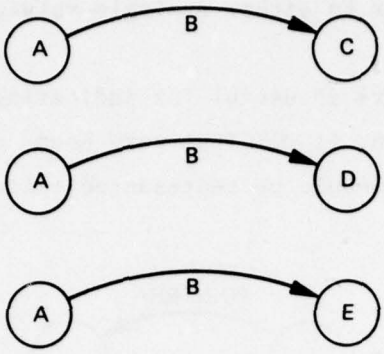
GIRS is a hashed-address associative memory scheme composed of 15 FORTRAN subroutines designed to accommodate the insertion, retrieval, modification, and deletion of information. GIRS enables data relationships to be expressed in an arbitrarily directed (plex) data structure known as a graph. After a graph has been created, GIRS can be called upon to store the graph on disk and fetch it from disk.

In GIRS, information is stored conceptually as a set of primitive structures called node-link-node triples:

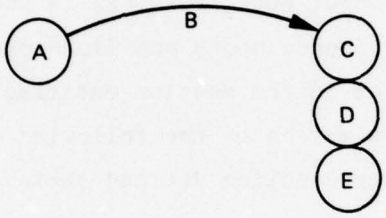


A Node-Link Node Triple

Links may be thought of as pointers, arcs, edges, associations, or functions. Nodes may also be thought of as beads or points or as arguments of those functions. The function in the triple A, B, C would be B(A). In this example, the node A is the source node (the argument) and the node (C) is the sink node (the value). Each triple represents a single relationship in which the source node is said to be associated with the sink node via a link. The collection of all of the relationships (triples) makes up the graph. Complex relationships can be expressed by combining these triples in various ways. For example, the triple can be a component of a list (also known as a multivalued list (MVL)),



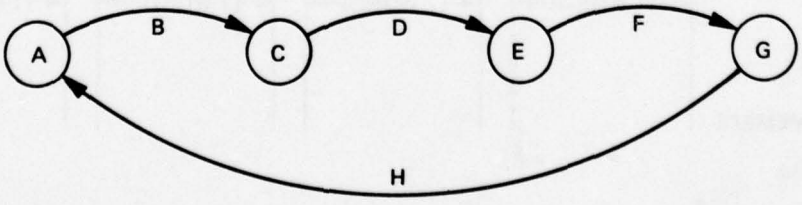
alternately drawn as



or it can be a component in a <u>string</u>

alternately drawn as

With the addition of the triple G,H,A this string would become a <u>circuit</u>.



A function can refer to either a single-valued list (SVL) or to an MVL.

A GIRS-type structure is useful for indicating relationships among data blocks. For example, if POINTERA were used to associate BLOCK1 with BLOCK2, the relationship would be represented conceptually by the structure

in which BLOCK1 is the source node, POINTERA is the link, and BLOCK2 is the sink node or value. Source nodes and links are represented by random numbers as discussed later in the section entitled "Initialization of the GIRS Buffer". Sink nodes may be of the following types:

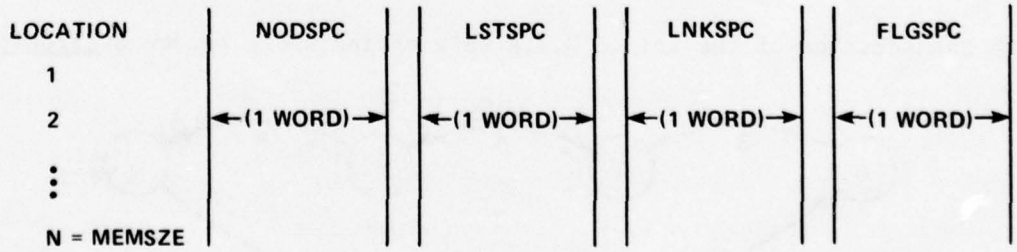
- o Random numbers representing defined variables
- o Integer data
- o Hollerith data

The latter two types must always be terminal points in the graph.

GIRS triples are stored in a buffer composed of four information fields. In general, one cell in each of these four fields is needed to store a triple. Each address refers to the same relative location in each field, for example:

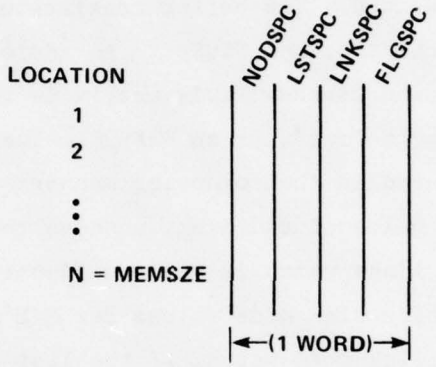
THE GIRS BUFFER

Unpacked WRKSPC



GIRS provides the CDC 6700 user with an option for making a time/space trade-off as concerns buffer space. In the arrangement just indicated (which is for the "unpacked" version), the fields exist as physically separate arrays of the same size (MEMSZE), each field taking up a full word. An option is available for packing the four fields into a single array in which each field takes up approximately one-fourth of a word as shown here:

PACKED WRKSPC



For the purposes of this report, the term WRKSPC always refers to all four fields collectively.

A special language is available to the user of GIRS which enables a one-to-one trace from graph operations to code. This language, called GIRL (Graph Information Retrieval Language),[2] facilitates the use of all the GIRS features described in this report. Since the code for this language is concise and readable, its use will result in reduced programming time. By embedding GIRL within FORTRAN via the GIRL preprocessor, the user gains both a graph symbol manipulation capability and a numeric processing capability. Note that two extra GIRS subroutines (detailed in Appendix C) are needed when GIRS is used in conjunction with the GIRL preprocessor.

The pages that follow explain computer representation of the graph structure and describe how GIRS subroutines are used.

COMPUTER REPRESENTATION OF THE GRAPH STRUCTURE

THE FOUR FIELDS OF WRKSPC

The triples which make up the graph must be stored in memory in such a way that operations on these triples can be efficiently performed. In GIRS, the buffer exists initially as a ring of available entry space, referred to as Available Space (AS). A special register, REGASP, (REGister of Available SPace), indicates which location in AS is to be used next. When an address within the buffer has been determined for a triple, that location is deleted from the list of available space and the ring of available space is reconnected. The buffer consists of a matrix of four fields - NODSPC, LSTSPC, LNKSPC, and FLGSPC. An address refers to the same location in each of the fields. This matrix is referred to as the GIRS buffer, or just, "the buffer", or as WRKSPC. The four fields of each location in AS are used in the following manner:

NODSPC contains the value of the link, used as the key. A key is that value from a set of links which is used to identify a particular function. NODSPC also holds sink node values for MVL's.

LSTSPC contains the sink-node value, if the list is single-valued (SVL), or a down pointer to the location holding the next value, if the list is multivalued.

LNKSPC is used to resolve address conflicts so that the location which contains the head of the function will point either to itself or to the location of a displaced function having the same computed address. As other conflicts occur at a particular location, a circular list for that location is formed, called a conflict list. LNKSPC is also used to store pointers to the preceding value on the MVL.

FLGSPC is composed of eight bits which describe the contents of NODSPC, LSTSPC, and LNKSPC at that location. The GIRS buffer will be described in greater detail in the following seven sections.

INITIALIZATION OF THE GIRS BUFFER

Initialization of the GIRS buffer is accomplished by calling Subroutine LVSETP. LVSETP arranges all the locations of the buffer in a circular "last-next" list. This list is formed initially by calling Subroutine LVGRN which returns a sequence of numbers $\sigma(X)$ that represents a

permutation of the set of numbers ranging from 1 through MEMSZE (the requested GIRS memory size). Next, the pointers in NODSPC are set to the i^{th}-1 number (which represents an address) in the returned sequence, and the pointers in LSTSPC are set to the i^{th}+1 number in that sequence. A special register called REGASP is set by LVSETP to point to the entry cell of AS. The pointers in the first and last addresses of the sequence point to each other. LNKSPC is zeroed out and each location in FLGSPC is set to FL3MSK, a flag which indicates whether a node or link has been assigned a random number which has the flagged location as its value. Initially, the AS is arranged as follows:

Location	Sequence	NODSPC	LSTSPC	LNKSPC	FLGSPC
1				0	20_8
2					
.	σ(x-1)	σ(x-2)	σ(x)	.	.
.	σ(x)	σ(x-1)	σ(x+1)	.	.
.	σ(x+1)	σ(x)	σ(x+2)	.	.
MEMSZE				0	20_8

If a triple is to be entered into the buffer at a computed address, that address must be removed from the AS list. This is done automatically by Subroutine LVUPDT (not user callable) which updates the pointers preceding and following the computed address in NODSPC and LSTSPC and updates REGASP to a new location in AS. For example, assume that a triple is to be placed into WRKSPC in location x. The buffer would then look as follows:

Location	Sequence	NODSPC	LSTSPC	LNKSPC	FLGSPC
1				0	20_8
2					
.	σ(x-1)	σ(x-2)	σ(x+1)	0	20_8
X		(Triple description)			
.	σ(x+1)	σ(x-1)	σ(x+2)	0	20_8
MEMSZE				0	20_8

REPRESENTATION OF NODES AND LINKS

Before nodes and links may be used in a graph, they must be "defined" by Subroutine LVGRN. This subroutine assigns a unique random number in the range 1 to MEMSZE to the node or link. The total number of nodes and links which the user may define may not exceed MEMSZE or the program will terminate.

A node or link which is defined by LVGRN may be used as a source node, a link, or a sink node. Although source nodes and links may be represented only by random numbers (modulus MEMSZE), sink nodes may contain the following kinds of data:

1) Random numbers (modulus MEMSZE) as returned by LVGRN
2) Integers
3) Hollerith character strings.

COMPUTATION OF A BUFFER ADDRESS FOR A FUNCTION

When a triple is to be placed into the buffer, its address is computed by GIRS on the basis of the internal values of the source node and link assigned to them by LVGRN. For the triple (A,B,C)--in which A is the source node, B is the link, and C is the sink node--the address would be computed as follows:

$$\text{Address} = \begin{cases} \text{MEMSZE, if } (A+B) \bmod (\text{MEMSZE}) = 0; \text{ otherwise} \\ (A+B) \bmod (\text{MEMSZE}) \end{cases}$$

For the same triple (A,B,C), the random number which represents the source node may be computed as follows:

$$A = (\text{Address} - \text{Link}) \bmod (\text{MEMSZE})$$

THE CONFLICT LIST

It is possible that the same address will be computed for more than one function. For example, consider the functions (A,B,C) and (B,A,C). Since a particular location in WRKSPC can accommodate only one function, another function subsequently given the same computed address must be placed in a different location (the next available location in AS) as determined by REGASP.

To enable access of such a displaced function, a circular conflict list is created in LNKSPC. The function actually residing in the originally computed address is considered to be the head of the conflict list. This is true even if there are no conflicts, in which case the function points to itself. During retrieval, the desired function in the conflict list is identified by the key (the link) in NODSPC.

To illustrate, let us insert the following triples

(A,B,C), (E,F,G), (X,Y,Z)

into the buffer such that

(A+B) Mod MEMSZE = (E+F) Mod MEMSZE = (X+Y) Mod MEMSZE = L

Although L is a computed address, M and N are assigned by REGASP.
If the three triples were placed into the buffer in the order listed, the conflict list and appropriate keys for these functions in WRKSPC would be as follows:

Location	NODSPC	LSTSPC	LNKSPC	FLGSPC
.				
.				
L	B		M	
.				
.				
M	F		N	
.				
.				
N	Y		L	

Note that the head of the conflict list, in this case (A,B,C) may displace any information which is not the head of a conflict list. Furthermore, if the triple (A,B,C) (which is the head of the conflict list by virtue of being placed into the buffer first) were to be deleted, the triple (E,F,G) would be transferred to location L, as indicated following:

Location	NODSPC	LSTSPC	LNKSPC	FLGSPC
L	F		N	
.				
.				
N	Y		L	

One final note. A time-versus-space tradeoff is of concern here. As the buffer fills, the average length of the conflict lists increases. Thus, the longer the conflict list, the greater the value retrieval time. Increasing the buffer size (MEMSZE) will reduce the number of hash-address collisions to be resolved.

THE FLAG FIELD

The flag field, contained in FLGSPC, consists of six one-bit flags and one two-bit flag:

FLGSPC

0	1	2	3	4	5	6-7

Each flag describes a different aspect of the contents of the associated location in WRKSPC (Table 1).

TABLE 1 - THE FLAG FIELD

Flag	Flag Value	Contents of Associated Location
Flag 0	2^7	Head of a multivalued list.
Flag 1	2^6	Location already occupied.
Flag 2	2^5	Either (1) a value on a multivalued list, or (2) the head of a multivalued list.
Flag 3	2^4	A node or link has been assigned a random number which has this location as its value.
Flag 4	2^3	The head of a multivalued list which has been modified either by an insertion or an indexed deletion, thus bypassing the "saved index" upon retrieval feature. (See the description of Subroutine LVFNV for further details.)
Flag 5	2^2	The head of a conflict list.
Flag 6-7	2^1+2^0	Type of value contained in the location: 00 Random number 01 Numeric data 10 Continuing string of Hollerith data 11 The only, or final, cell in a Hollerith data string

THE SINGLE-VALUED FUNCTION

After the buffer location for the function has been determined, the function is placed in WRKSPC, with the key (which is the link) in NODSPC, and the value in LSTSPC. It is not necessary to store the value of the source node, since it can be recovered by the equation

Source node = (location - link) Mod MEMSZE

where "location" is the address of the head of the conflict list for that function. The conflict list pointer is placed into LNKSPC. If the function is the first to claim a particular location, the address of that location will be the value in LNKSPC. The format of an SVL as entered into WRKSPC at location L = [(source node + link) Mod MEMSZE] is as follows:

Location	NODSPC	LSTSPC	LNKSPC	FLGSPC 0 1 2 3 4 5 6-7 (flag descriptors)
L	Key (link)	Value	Conflict List Pointer	

To illustrate, assume that the triple (A,B,C) is to be placed into the buffer. If C were a random number, and A+B were equal to Locl, and Locl has been in available space previously, the address Locl in WRKSPC would appear as follows:

Location	NODSPC	LSTSPC	LNKSPC	FLGSPC 0 1 2 3 4 5 6-7
Locl	B	C	Locl	0 1 0 1 1 0 0

The flags of FLGSPC provide the following information:

Location L is not in AS

Location L contains an SVL

THE MULTIVALUED FUNCTION

Multivalued functions require one location of WRKSPC for each value in the list plus one extra location for overhead. In the overhead location, which is called the head of the MVL, NODSPC will contain the key (link), just as was true in the SVL. However, LSTSPC will contain a down

pointer to the first value on the list. LNKSPC will contain the conflict list pointer. The location within WRKSPC for the head of the MVL is determined in the same way as for the SVL. The locations of the values to be stored are determined by REGASP.

The four fields at these locations will contain the following information:

NODSPC	Value of sink node
LSTSPC	Down pointers to the next value on the list; for the last value on the list, LSTSPC will contain a pointer back to the head of the MVL
LNKSPC	An up-pointer to the previous value on the list (except for the first value in the list in which LNKSPC contains a pointer to the last value in the list).

Therefore, the entries of the MVL will be linked together in an up-down circular fashion, in the following way:

Location	NODSPC	LSTSPC	LNKSPC	FLGSPC 0	1	2	3	4	5	6-7
Loc_1	Key (link)	Pointer to first value (in Loc_2)	Conflict list pointer	1	1	1			1	
Loc_2	Value 1	Down pointer to next value	Pointer to last value	0	1	1		0	0	
Loc_{n+1}	Value n	Pointer to head of MVL- Loc_1	Pointer previous value in Loc_{n-1}	0	1	1		0	0	

Recall that the buffer address for the MVL head is determined as follows: Loc_1 = (source node + link) Mod MEMSZE. The Loc_i have been removed from AS at the direction of REGASP.

Accessing either end of a MVL is a simple matter:

Location of first value = LSTSPC (Loc_1)

Location of last value = LNKSPC (LSTSPC (Loc_1))

Suppose that we wish to insert a multivalued function consisting of the triples

(E,F,G), (E,F,H), and (E,F,J)

where G,H and J are integers, and Loc = (E+F) Mod MEMSZE. If Loc had been previously contained in AS, the function would be stored in WRKSPC as follows:

Location	NODSPC	LSTSPC	LNKSPC	FLGSPC 0 1 2 3 4 5 6-7
Loc	F	L	Loc	1 1 1 1 1 0 0
•				
•				
L	G	M	N	0 1 1 0 0 0 1
•				
•				
M	H	N	L	0 1 1 0 0 0 1
•				
•				
N	J	Loc	M	0 1 1 0 0 0 1

where locations L, M and N were subsequently removed from AS.

If, however, Loc had not been in AS previously, but instead had been used to contain the triple (A,B,C) such that

loc = (A+B) Mod MEMSZE = (E+F) Mod MEMSZE

a conflict list would result and the function would be stored in the buffer as follows:

Location	NODSPC	LSTSPC	LNKSPC	FLGSPC 0 1 2 3 4 5 6-7
Loc	B	C	K	0 1 0 1 1 0 0
•				
•				
K	F	L	Loc	1 1 1 1 0 0 0
•				
•				
L	G	M	N	0 1 1 0 0 0 1
•				
•				
M	H	N	L	0 1 1 0 0 0 1
•				
•				
N	J	K	M	0 1 1 0 0 0 1

where locations K, L, M, and N have been removed from AS at the direction of REGASP.

Note that conflict lists are needed to resolve address collisions of functions, whereas multivalued lists are needed to accommodate functions that have more than one value.

HOLLERITH DATA

One form of data which GIRS handles is the Hollerith character sequence. The maximum number of characters which may be stored for each triple depends on the host machine being used, and on whether the "packed" or the "unpacked" version of GIRS is being used. Data must be entered in a right-adjusted ("R") format for the packed version. The maximum number of characters allowed per triple for each machine on which GIRS is now operable is indicated in Table 2.

TABLE 2 - MAXIMUM NUMBER OF HOLLERITH CHARACTERS ALLOWED PER TRIPLE

CDC 6700	Unpacked version	$\leq$10 characters
	Packed version	$\leq$ 3 characters
UNIVAC 1108		$\leq$ 6 characters
PDP 11/45		$\leq$ 2 characters

INTEGER DATA

GIRS also handles integer values. The maximum size of value to be stored again depends on the word size of the host machine and on the version of GIRS (packed or unpacked) used. Table 3 indicates the number of maximum values that may be stored for each computer.

TABLE 3 - MAXIMUM SIZE OF INTEGER VALUES THAT MAY BE STORED PER TRIPLE

CDC 6700	Unpacked version	$n \leq 2^{59} - 1$
	Packed version	$n \leq 2^{17} - 1$
UNIVAC 1108		$n \leq 2^{35} - 1$
PDP 11/45		$n \leq 2^{15} - 1$

In the packed mode, the most significant bits of values having a magnitude greater than 2^{17} -1 will be truncated.

SAMPLE GIRS STRUCTURE

The following example shows the appearance of the buffer in successive stages following insertions. For this example, we will use the unpacked version of GIRS as it is implemented on the CDC 6700 computing system. The importance of time order, as it affects both conflict and multivalue lists, is demonstrated. The triples are inserted in the following order (possibly with other GIRS actions interspersed):

(A,B,C)

(X,Y,Z)

(A,B, "HOL DATA")

(A,B,X)

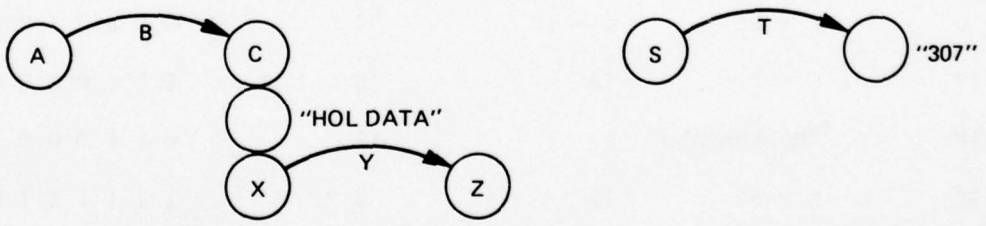
(S,T, "307")

Subroutine LVGRN has assigned random numbers to the nodes and links as follows:

A = 33	B = 66	C = 2	X = 100	Z = 10
S = 40	T = 19	Y = 99	MEMSZE = 100	

Note that FLAG3 in FLGSPC will be turned on for WRKSPC locations 2, 10, 19, 33. 40, 66, 99, and 100.

The GIRS structure for this information is as follows:



The contents of the buffer after each of the five insertions are indicated as follows:

Location	NODSPC	LSTSPC	LNKSPC	FLGSPC 0 1 2 3 4 5 6-7
REGASP = 6				
1. (A,B,C)				
1				
.				
.				
.				
99	B = 66	C = 2	99	0 1 0 1 1 1 0 0
100				
2. (X,Y,Z)				
1				
.				
.				
.				
6	Y = 99	Z = 10	99	0 1 0 0 1 0 0 0
.				
.				
.				
99	B = 66	C = 2	6	0 1 0 1 1 1 0 0
100				
REGASP = 17				
3. (A,B, "HOL DATA")				
1				
6	Y = 99	Z = 10	99	0 1 0 0 1 0 0 0
17	C = 2	18	18	0 1 1 0 0 0 0 0
18	"HOLbDATAbb"	99	17	0 1 1 0 0 0 1 1
99	B = 66	17	6	1 1 1 1 1 1 0 0
100				

Location	NODSPC	LSTSPC	LNKSPC	FLGSPC 0 1 2 3 4 5 6-7
REGASP = 59				
4. (A,B,X)				
1				
6	Y = 99	Z = 10	99	0 1 0 0 1 0 0 0
17	C = 2	18	59	0 1 1 0 0 0 0 0
18	HOLbDATAbb	59	17	0 1 1 0 0 0 1 1
59	X = 100	99	18	0 1 1 0 0 0 0 0
99	B = 66	17	6	1 1 1 1 1 1 0 0
100				
REGASP = 60				
5. (S,T, "307")				
1				
6	Y = 99	Z = 10	99	0 1 0 0 1 0 0 0
17	C = 2	18	60	0 1 1 0 0 0 0 0
18	"HOLbDATAbb"	60	17	0 1 1 0 0 0 1 1
59	T = 19	"307"	59	0 1 0 0 1 1 0 1
60	X = 100	99	18	0 1 1 0 0 0 0 0
99	B = 66	17	6	1 1 1 1 1 1 0 0
100				

USE OF GIRS SUBROUTINES

INITIALIZATION

Operating Sequences

To execute a GIRS program, labeled commons must be set up in the driving program and the buffer, and the random number generator (Subroutine LVGRN) must be either initialized by Subroutine LVSETP or restarted from disk (Subroutine LVFECH).

The letters "LV" must not be used to begin subroutine and variable names. These initial letters are reserved for GIRS.

The following common blocks must be included in the user's main routine:

/LVARGS/,/LVRAND/,/LVTABL/,/LVVSEQ/

If packed version of GIRS is to be used, the common blocks /LVVTR1/ and /LVVTR5/ must be added. A program to be run using the unpacked version of GIRS or the PDP-11 version requires additional blocks:

/LVVTR1/,/LVVTR2/,/LVVTR3/,/LVVTR4/,/LVVTR5/,/LVVTR6/,/LVVTR7/,/LVVTR8/

The section on labeled commons contains the format of each block.

The user must first decide on a size for the GIRS buffer (MEMSZE, an input parameter to LVSETP). Enough space must be alloted to accommodate all the triples to be inserted into the graph. Each single-valued function requires one location; each multivalued function requires one location per value plus one location per function for overhead. In determining the buffer size, the user should be aware that the lower the ratio of spaces occupied by triples to AS, the shorter the average length of the conflict lists and, hence, the shorter the average retrieval time. MEMSZE also determines the maximum number of nodes and links which may be defined.

In addition to supplying input parameters to LVSETP, the user must also set certain variables from the labeled commons:

(In COMMON/LVARGS/)

NVAL = 1

IPOS = 1

ITYP = 3

NSKIP = 0 or 1 (See the section on Retrieval of Values)

INDXON = 0 (PDP-11 implementation only)

(In COMMON/LVVTRS)

BINFIL = the logical file number from the ASSIGN call or PROGRAM card if a compression/expansion of the GIRS Buffer Space is to be executed; otherwise 0.

Finally, for both the PDP-11 implementation and the unpacked version, the integer arrays NODSPC, LSTSPC, LNKSPC, and FLGSPC from the labeled commons /LVVTR1/,/LVVTR2/,/LVVTR3/, and /LVVTR4/, respectively, must be dimensioned to MEMSZE.

There are four types of runs involving GIRS:

- o Creation of a GIRS graph structure.
- o Operation on a previously created graph structure
- o Compression or expansion of the GIRS buffer which contains the graph structure.
- o Packing of the four information fields of the four arrays into a single array, or vice versa. This procedure is needed when a graph created in one version of GIRS must be used in another version. (CDC implementation only).

The common blocks needed in each case are indicated in Appendix F, "Variables in Labeled Common." The steps to be performed for the different types of runs are as follows:

Creation Run

1. Define logical unit number (lun) in PROGRAM card.
2. Define MEMSZE to be the length of the GIRS buffer.
3. Initialize all variables mentioned previously in this section.
4. Dimension SEQSPC (1) [From labeled common /LVVSEQ/].

5a. Dimension WRKSPC (GIRS buffer size)* [Packed version]

5b. Dimension NODSPC (GIRS buffer size) [Unpacked version and PDP-11 implementation]

Dimension LNKSPC (GIRS buffer size) [Unpacked version and PDP-11 implementation]

Dimension FLGSPC (GIRS buffer size) [Unpacked version and PDP-11 implementation]

---

*Equals MEMSZE.

6. Set KOMPAN to 0, MAPSZE to 1 [From labeled commons/LVVTR5/ and /LVTABL/]

7. Set KPRIME to the first prime number $\geq 1/2 \sqrt{\text{MEMSZE}}$ [From common /LVRAND/]

8. CALL LVSETP

9. Define all nodes and links via calls to LVGRN

.
.
.

(Body of Program)

.
.
.

10. Define logical unit number (lun) in a CALL ASSIGN card*

[At the end of the program, a call to subroutine LVDUMP will save the graph. See section entitled "Disk Storage and Retrieval of a Graph" for further details.]

11. CALL LVDUMP (0,0,lun)

12. Write out the values of all nodes and links to be used in a later run.

Production Run (Operating on a previously created graph structure)

1. Define a logical unit number (lun) in PROGRAM card.**

2. Define MEMSZE to be the length of the GIRS buffer.

3. Initialize all variables mentioned previously in this section.

4. Dimension SEQSPC (1)

5a. Dimension WRKSPC (GIRS buffer size) [Packed version]

5b. Dimension NODSPC (GIRS buffer size) [Unpacked version and PDP-11 implementation]

Dimension LSTSPC (GIRS buffer size) [Unpacked version and PDP-11 implementation]

Dimension LNKSPC (GIRS buffer size) [Unpacked version and PDP-11 implementation]

Dimension FLGSPC (GIRS buffer size) [Unpacked version and PDP-11 implementation]

---

*PDP-11 implementation only.

**CDC implementation only.

6. Set KOMPAN to 0, MAPSZE to 1.
7. Define logical unit number (lun) in a CALL ASSIGN card*
8. CALL LVFECH(lun)
9. Read in the saved values or previously created nodes and links needed to operate on the graph.
10. Define all new nodes and links via calls to LVGRN.

(Body of Program)

Compression/Expansion Run

(All versions)

1. Define logical unit number (lun) in PROGRAM card.*
2. Define MEMSZE to be the length of the GIRS buffer. Set MEMSZE to the new buffer size. If the buffer size is to be minimized, set MEMSZE to 1.
3. Initialize all variables mentioned previously in this section.
4. Dimension SEQSPC (1)
5. Define KOMPAN:
   - = 1 Regular compression/expansion run
   - = 2 Compress buffer to minimum length plus IEXTRA number of free spaces
6. Define BINFIL (logical unit number)
7. Define MAPSZE to old GIRS buffer size or new buffer size, whichever, is larger.
8. Dimension MAP (mapsze)
9. Define IEXTRA if the value of KOMPAN is 2

(Packed version only)

10. Dimension WRKSPC (new GIRS buffer size)
11. Dimension WORKSP (old GIRS buffer size)
12. CALL LVSETP
13. CALL LVDUMP (0,0,lun)
14. Write out the values of all nodes and links to be used in a later run.

---

*CDC implementation only.

(Unpacked version only)

10. Dimension NODSPC (new GIRS buffer size)
    Dimension LSTSPC (new GIRS buffer size)
    Dimension LNKSPC (new GIRS buffer size)
    Dimension FLGSPC (new GIRS buffer size)
11. Dimension NODESP (old GIRS buffer size)
    Dimension LISTSP (old GIRS buffer size)
    Dimension LINKSP (old GIRS buffer size)
    Dimension FLAGSP (old GIRS buffer size)
12. CALL LVSETP
13. CALL LVDUMP (0,0,lun)
14. Write out the values of all nodes and links to be used in a later run.

Pack/Unpack Run

(Both Versions)

1. Define logical unit number (lun) in PROGRAM card.*
2. Define MEMSZE to be the length of the GIRS buffer. For a compression or expansion run, set MEMSZE to the new buffer size. If the buffer size is to be minimized, set MEMSZE to 1.
3. Initialize all variables mentioned previously in this section.
4. Dimension SEQSPC (1).
5. Set KOMPAN = 0, MAPSZE = 1.
6. Define logical unit number (lun) in a CALL ASSIGN card**

(Packed version only)

(to change GIRS buffer from four arrays to one array)

7. Dimension NODSPC (GIRS buffer size)
   Dimension LSTSPC (GIRS buffer size)
   Dimension LNKSPC (GIRS buffer size)
8. CALL LVPACK (NODSPC, LSTSPC, LNKSPC, lun)

---

*CDC implementation only.

**PDP-11 implementation only.

(Unpacked version only)

(to change GIRS buffer from one array to four arrays)

7. CALL LVUNPK (lun)

See section on Deck Setups and Command Sequences for further details on initialization.

Descriptions of the two subroutines used for initialization, LVSETP and LVGRN, follow on pages 24 and 26 respectively.

Subroutine LVSETP

Function: Initializes the four fields in the GIRS buffer and those variables needed for Subroutine LVGRN; initializes the REGister of Available SPace (REGASP) and determines whether a particular run is to compress or expand the buffer.

Calling Format:

CALL LVSETP

Input Parameters:

(In COMMON/LVVTR1/)

MEMSZE Length of the GIRS buffer. It must be judiciously chosen to balance the length of conflict lists with the associated value retrieval time. See section on Conflict Lists for further details.

(In COMMON/LVVTR5/)

KOMPAN Value which determines whether the run will be normal or one in which the GIRS buffer is either compressed of expanded:

= 0 Normal run

= 1 Compression or expansion run to MEMSZE

= 2 Compress buffer to a minimum size plus IEXTRA free spaces

(In COMMON/LVRAND/)

KPRIME First prime number $\geq 1/2 \sqrt{\text{MEMSZE}}$

Output Parameters:

The following parameters are used internally by GIRS.

(In COMMON/LVVTR1/)

REGASP - Register of available space.

(In COMMON/LVRAND/)

KSEED NDNODE

NROW NDROW

KTEMP

Comments:

LVSETP must be the first GIRS Subroutine called if a GIRS memory is to be created. If a previously created graph is to be used, Subroutine LVFECH is the first one called.

Astract:

First LVSETP arranges the entire GIRS buffer into an ordered list of available space. NODSPC and LSTSPC will have a "last" or a "next" location value, respectively. The sequence of the "next" (and, of course, "last") locations is determined by MEMSZE number of calls to LVGRN. The first of these calls determines the initial value of REGASP.

LVSETP then reinitializes the variables in labeled common /LVRAND/ so that LVGRN may return random values in the same sequence as before. Finally, LVSETP tests KOMPAN to determine whether or not the buffer will be compressed or expanded.

Program Length:

CDC 6700		PDP-11
Unpacked Version	Packed Version	
47_8 (39)	74_8 (60)	222_8 (246)

Subroutines Called:

LVCMPN

LVGRN

LVLFSH (packed version)

Subroutine LVGRN

Function: Assigns a unique "random" number to a given GIRS identifier.

Calling Format:

Call LVGRN (NODE)

Input Parameters:

The following system input parameters are provided by either LVSETP or LVFECH

(In COMMON/LVVTR1/)

MEMSZE Length of the GIRS buffer.

(In COMMON/LVRAND/)

KPRIME NDNODE

KSEED NDROW

KROW KTEMP

Output Parameters:

(Formal Argument)

NODE Random integer generated by LVGRN.

(In COMMON/LVRAND/)

NROW

NDNODE (Updated internally)

NDROW

KTEMP

Comments:

LVGRN generates a different integer in the range 1 to MEMSZE each time it is called. An attempt to define more than MEMSZE number of identifiers will terminate the program unless a random number has been "undefined" by Subroutine LVRTRN.

Equivalent GIRL Code:

Identifiers may be defined in GIRL in two ways. At the beginning of each routine, a list of identifiers may be defined in the following manner:

```
G     DEFINE NODE1,..., NODEn
```

Identifiers (which must be integers) may be given random numbers at any time with the following code:

```
G     $'NODE1
```

Abstract:

LVGRN generates a complete and repeatable sequence of "random" numbers of the range 1 to MEMSZE. This sequence may be modified by changing the prime number, KSEED, and, of course, MEMSZE. The calculation for the optimum prime number

$$p \cong 1/2 \sqrt{\text{MEMSZE}}$$

has been determined by Berkowitz[1] (pp. 18-25).

The purpose of providing a predictable "random" sequence is to minimize the number of early address collisions, thus keeping the lengths of the conflict lists as short as possible. A description of the algorithm from Berkowitz[1] is repeated here:

> "Suppose we first generate P [=KPRIME] numbers in the generator s_1, for P a prime, as just described. Call these numbers $s_1, s_2, \ldots, s_p$, where $s_j = js_1 \pmod P$. Then let each s_j serve as a further generator of a sequence
>
> $$\{s_{jh} \mid s_{jh} \leq M,\ s_{jh} = s_j + h^2/2 + (P-1/2)h - P\} \qquad (4)$$
>
> The sequence in Equation (4) is in fact generated recursively using only additions, as we shall see. As shown [below] the sequence thus far generated leaves certain residues..."

Node Generator Row Sequence

increasing increment

s_1	s_1+P+1	s_1+2P+3	...	initial generation
s_2	s_2+P+1	s_2+2P+3	...	
.	.	.		
.	.	.		
.	.	.		
s_p	s_p+P+1	s_p+2P+3	...	
P+1	2P+2	3P+5	...	residues
	2P+3	3P+5	...	
		3P+6	...	

The flow chart for generating this sequence is given in Appendix D.

To illustrate the concept just discussed in the excerpt from Berkowitz, let us choose MEMSZE=100. KPRIME is chosen to be the prime number nearest to:

$$1/2 \sqrt{\text{MEMSZE}} = 5$$

and KSEED = KPRIME/2 = 2. The sequence of numbers would be generated as follows:

2	8	15	23	32	42	53	65	78	92
4	10	17	25	34	44	55	67	80	94
1	7	14	22	31	41	52	64	77	91
3	9	16	24	33	43	54	66	79	93
5	11	18	26	35	45	56	68	81	95
6	12	19	27	36	46	57	69	82	96
	13	20	28	37	47	58	70	83	97
		21	29	38	48	59	71	84	98
			30	39	49	60	72	85	99
				40	50	61	73	86	100
					51	62	74	87	
						63	75	88	
							76	89	
								90	

Program Length:

CDC 6700		PDP-11
Unpacked	Packed	
60_8 (48)	60_8 (48)	247_8 (167)

Called by the Following Subroutines:

LVCMPN and LVSETP

RETRIEVAL OF VALUES

Subroutine LVFIND is, in a sense, the heart of GIRS, since it performs the function address computation and function search for the retrieval and manipulation routines. A call to LVFIND must precede all calls to Subroutines LVFNV, LVDLET, and LVNSRT. LVFNV must also be called if an indexed insertion or indexed deletion is to be performed. The two most important labeled commons used with the basic routines (LVFIND, LVFNV, LVNSRT and LVDLET) are /LVARGS/ and /LVADDR/. The labeled common /LVARGS/ must be included in the driving program. Descriptions of the two routines LVFIND and LVFNV follow:

Subroutine LVFIND

Function: Computes the potential function address and determines whether or not the function exists. If it does, determines
(1) its location within the buffer (since it may not be first on the conflict list, and may therefore reside anywhere in WRKSPC)
(2) whether it is an SVL or MVL
(3) buffer location of preceding function on the conflict list
(4) location of the first value of the function. Returns that first value.

Calling Format:

CALL LVFIND

Input Parameters:

(In COMMON/LVARGS/)

IFUNC Link of the triple, also known as the function. It must be a random number as defined by LVGRN.

IARG Source node of the triple, also known as the argument of the function. It must be a random number as defined by LVGRN.

Output Parameters:

(In COMMON/LVADDR/)

IADD Computed function address

THIS Location of the function on the conflict list.

LSTHED = -1 SVL found

= 0 No list found

> 0 MVL found, location of first value is returned.

LOC Location of the first (or only) value of the function.

= THIS (If an SVL has been found)

= LSTHED (If an MVL has been found)

LAST Location of the preceding function on the conflict list.

If the retrieved function is the head of the conflict list, LAST is undefined.

(In COMMON/LVARGS/)

IPOS Set to 1 (default value)

ITYP Set to 3 (default value)*

IVAL Retrieved value. If the function does not exist, IVAL is set to IARG (the source node).

ITESTR = 1 Function exists

= -1 Function does not exist

Comments:

The prime concern in the use of LVFIND is that both IARG (the source node) and IFUNC (the link) be well defined integers. They must have been assigned random numbers previously by Subroutine LVGRN.

The first value of the function is returned, no matter what type it is.

Abstract:

The first task performed by LVFIND is to compute the function address as described in the section "Computation of a Buffer Address for a Function." (Due to the inadequate diagnostics of the PDP-11 computer, the PDP-11 implementation of LVFIND includes an extra test to make sure that the address (IADD) is $\leq$ MEMSZE). A search for that function is then started. If a function that is the head of a conflict list is not found at that location, retrieval failure is indicated, default values are set, and the value of the source node is stored in the "value" parameter.

---

*See description of Subroutine LVFNV for explanation and list of value types.

If a function that is the head of a conflict list is found, tha list will be traversed and the key in NODSPC compared to the requested link of the triple. If no match is found when the search finishes at the head of the conflict list, the failure exit is taken.

If a match is found, meaning that the function already exists within the buffer, the value in THIS is set to the location in which the function is actually found. The next test determines whether the list is an SVL or MVL. The value (sink node) is then retrieved from either LSTSPC (SVL) or NODSPC (MVL) and LOC, LSTHED, IPOS, and ITYP are then set prior to returning to the calling routine.

Equivalent GIRL Code:

G NODE+LINK

Program Length:

CDC 6700		PDP-11
Unpacked Version	Packed Version	
41_8 (33)	104_8 (68)	327_8 (215)

Subroutines Called:

LVRTSH (packed version)

Called by the Following Subroutines:

LVDLET LVNSRT LVCMPN

Subroutine LVFNV

Function:

Retrieves the IPOSth value of the type indicated from the top or bottom (depending on the sign of IPOS) of a list of values of a specified function.

Calling Format:

CALL LVFNV(INDEX,INDXAD,KFUNC,KARG) PDP-11 version

CALL LVFNV(INDEXS) All other versions

Input Parameters:

(In COMMON/LVARGS/)

IFUNC Link of the triple, also known as the function. The value in IFUNC must be a random number as defined by a call to LVGRN. It does not have to be reset after the call to LVFIND.

IARG Source node of the triple, also known as the argument of the function. The value in IARG must be a random number as defined by LVGRN. It does not have to be reset after the call to LVFIND.

IPOS Position in the multivalue list, IPOS locations from the top (if IPOS is positive) or from the bottom (if IPOS is negative). If ITYP is specified, only that type of value is considered in determining the position.

ITYP Type of value to be retrieved:

= 0 Random Number

= 1 Integer data

= 2 Hollerith data

= 3 No specified type (default value)

ITYP is preset to 3 in LVFIND, so must be changed only if a specific value type is desired.

NSKIP Saved-index defeat switch. If NSKIP = 1, the saved-index operation (described on page 34) is skipped; otherwise the saved-index feature is in effect. NSKIP can be set either at the start of the program or just before a call to LVFNV (after which it may be reset).

ITESTR Indicator of whether or not the function exists. It is provided by LVFIND (which must precede a call to LVFNV).

(In COMMON/LVADDR/)

THIS Location of the function on the conflict list as returned by LVFIND.

LSTHED Indicator of SVL or MVL as returned by LVFIND.

LOC Location of the first value of the function.

(Formal Parameter set)

The formal parameter set is needed by LVFNV when the saved-index option is to be used. The parameter set consists of four variables, each of which must be unique for each new call to LVFNV involving a saved index. The four variables are packed into one word (INDEXS), except that for the PDP-11 implementation they are separate words.

KARG Source node associated with a particular call to LVFNV.

KFUNC Link associated with a particular call to LVFNV.

INDEX Position in the list of the value retrieved from the most recent call to LVFNV. If INDEX is negative, it is the position from the bottom of the list.

INDXAD Location in WRKSPC of the value retrieved from the most recent call to LVFNV.

If the saved-index option is not to be used (NSKIP=1), the parameter should be set to zero. For example,

```
NSKIP = 1
K = 0
CALL LVFNV(K)
```

(For the PDP-11 Implementation)

```
NSKIP = 1
K = 0
CALL LVFNV(K,K,K,K)
```

If the saved-index option is to be used (NSKIP=0), the parameter set for LVFNV must contain a unique integer variable for each separate call to LVFNV. The parameter set must be initialized to zero at the start of the program and should remain unchanged by the user routine.

For example,

```
     DATA I/0/
     NSKIP = 0
     DO 10 M = 1, 100
     CALL LVFIND
     IPOS = M
     CALL LVFNV(I)
  10 ARRAY (M) = IVAL
```

(For the PDP-11 implementation)

```
     DATA I,J,K,L/4*0/
     NSKIP = 0
     DO 10 M = 1, 100
     CALL LVFIND
     IPOS = M
     CALL LVFNV(I,J,K,L)
  10 ARRAY (M) = IVAL
```

Output Parameters:

(In COMMON/LVARGS/)

ITYP	Set to 3 (default value)
IVAL	Retrieved value ($IPOS^{th}$ value of the type ITYP). IVAL is set to IARG if the value cannot be found.
ITESTR	If the $IPOS^{th}$ value of the type ITYP exists, ITESTR = 1, otherwise ITESTR = -1.

(In COMMON/LVADDR/)

LOC	Location in WRKSPC of the $IPOS^{th}$ value of the type ITYP.

(Formal Parameter Set)

KARG	Set internally to IARG
KFUNC	Set internally to IFUNC
INDEX	Set internally to IPOS
INDXAD	Set internally to LOC

Comments:

If LVFNV is to be used, LVFIND must first be called, then IPOS must be set. ITYP should remain unchanged if it is not needed since a retrieval takes less time when ITYP is left to default.

If the saved-index option of LVFNV is not desired, set NSKIP = 1 (COMMON/LVARGS/) and set a dummy parameter equal to 0. The saved-index option may be turned on or off for each new call to LVFNV. If the

saved-index option is used, greater speed is achieved for repeated access to lists longer than two elements; however, some added precautions must be taken.

Saved-Index Facility: LVFNV traverses multivalue lists sequentially; thus N calls to LVFNV to access the first through the n^{th} items on a multivalue list result in N(N+1)/2 accesses to main memory. The saved-index facility reduces this number of accesses to N at the cost of one word* of main memory for every call to LVFNV. A different variable should be used for every call to LVFNV in which the saved-index option is used. Arguments to LVFNV must be initialized to zero at the beginning of the program and must not be changed thereafter by the user program.

CAVEAT: The saved-index facility is used to speed repeated accesses to indexed positions in a long multivalue list. It relies on the structure of the list remaining unchanged while saved-index is in effect. The saved-index operation may yield incorrect answers when a function is retrieved by two separate calls to LVFNV followed by an indexed deletion or insertion to the same function. If the program loops back to the two calls to LVFNV, the second call to LVFNV may result in an incorrect answer. To avoid such a possibility, the saved-index defeat switch (NSKIP) should be set to 1 before making the second call to LVFNV, and be reset to 0 immediately after the call. Also, ITYP must not be changed.

Abstract:

LVFNV first tests the output of LVFIND to determine whether or not the function exists. If it does not, control returns to the calling program. If it does, the function is then tested to determine whether it is an SVL or an MVL. If it is an SVL, the requested index (IPOS) must be ± 1 and the requested value type (ITYP) must be either the same as the sink node value type or unspecified. If these conditions are not met, the failure exit is taken.

Once it has been determined if an MVL exists, several tests are made, all but the first one related to the saved-index option. If any of the following questions 2 through 8 are answered affirmatively, the search

---

*Four words in the PDP-11 implementation.

must begin from the top or bottom of the MVL (according to the sign of IPOS), since the saved-index option cannot be used. Note that before question 4 can be asked, formal parameters KARG, KFUNC, INDEX, and INDXAD must be unpacked from the INDEXS.

1) If the first value is requested, a success exit may be taken, since the value will already have been retrieved by LVFIND.
2) Has the saved-index operation been switched off?
3) Have the formal parameters been set to zero?
4) Has the function been modified recently?
5) Have either the source node or the link been changed?
6) With respect to this call to LVFNV, has the direction of the search (the sign of IPOS) changed from that of the previous search of this function?
7) Has the value pointed to by the saved-index been moved to a new location in WRKSPC to make room for the head of a conflict list?
8) Would the search take less time if it began at the top (or bottom) of the list instead of at the location of the saved-index? In other words, is the value at the location of the requested index closer to the beginning (or end) of the list than to the position of the saved-index?

If the search starts at the top of the list, the list is traversed via the pointers in LSTSPC, (LOC = LSTSPC(LOC)), and the value type (if specified) is tested and the values are counted to see whether or not IPOS values have been passed. A failure exit is taken if the search ends at the top of the list. If the search starts from the bottom of the list, the list is traversed via the pointers in LNKSPC, (LOC = LNKSPC(LOC)).

If the saved-index feature can be used, the relative distance between the requested index and the saved index is computed and the direction of the search is determined. The list is then traversed as described in the preceding paragraph.

Upon a successful retrieval, the four variables of the parameter set are packed into INDEXS, and ITYP is set to the default value.

Equivalent GIRL Code:

G NODE+LINK.tsJ

where t is the type of value to be retrieved:

= ' Identifier (node defined by LVGRN)

= . Integer value

= / Hollerith value

= "blank" Any type value

s is the indicating direction of search:

= + or

"blank" Search from top of list

= - Search from bottom of list

J is the same as IPOS

Program Length:

CDC 6700		PDP-11
Unpacked Version	Packed Version	
248_8 (163)	272_8 (187)	705_8 (453)

Subroutines Called:

LVRTSH and LVLFSH

Called by:

LVNSRT and LVINCL

RETRIEVAL OF MVL INDEX OF GIVEN VALUE
OF A FUNCTION (INCLUSION)

Subroutine LVINCL

Function:

Determine the first MVL position of a given value.

Calling Format:

Call LVINCL

Input Parameters:

(In COMMON/LVARGS/)

INCLUD	Value on which the list position is to be searched.
IVAL	First value on the list by LVFIND
ITESTR	Set by LVFIND; denotes existence of the function.

(In COMMON/LVADDR/)

LSTHED	Set by LVFIND; denotes SVL or MVL.

Output Parameters:

(In COMMON/LVARGS/)

IPOS	First position in the MVL in which the indicated value is found.
INCLUD	1 Desired value has been found on the MVL.
	-1 Desired value has not been found on the MVL.

Comments:

Before LVINCL is called, LVFIND must be called with all its input parameters set. Also, INCLUD (from COMMON/LVARGS/) must be set to the value on which the search is to be performed. Except for IPOS, output is the same as if only LVFIND had been called.

Abstract:

First, ITESTR, as provided by LVFIND, is checked to determine whether or not the function exists. If it does not, a failure exit is taken, setting INCLUD = -1. Since LVFIND automatically returns the first value on a list, the next test is to determine whether the value matches that requested in INCLUD. If so, a success exit is taken, setting INCLUD = 1. The next test is to determine whether the function is a SVL or an MVL.

If an SVL, a failure exit is taken, since the only value on the list has already been examined. At this point, the search down the list begins, comparing IVAL (from LVFNV) and INCLUD. If the value is not found, a failure exit is taken, and IPOS is set to a value one greater than the number of values on the MVL. IVAL is reset to the first value on the list, regardless of success or failure, since output is the same as though only LVFIND had been called. If the value is found, INCLUD is set to 1.

Equivalent GIRL code:

Use of the GIRL inclusion operator can best be explained with three examples. Further discussions and examples are given in Berkowitz.[2]

Assume for all examples that the source node is NODE and the link is LINK:

Example 1. Delete value3 on the MVL

```
G    NODE+LINK-.: value3
```

Example 2. Determine the position of valuei (if such a value exists) on the MVL and name it INDEX; otherwise transfer to fail.

```
G    NODE+LINK: valuei/fail':INDEX
```

Example 3. Replace value1 on the MVL with value2.

```
G    NODE LINK: value1 value2
```

Program Length:

CDC 6700		PDP-11
Unpacked	Packed	
41_8 (33)	41_8 (33)	153_8 (107)

Subroutine Called:

LVFNV

INSERTION

Subroutine LVNSRT

Function:

Places the triple into the graph.

Calling Format:

CALL LVNSRT [for all types of insertions on the PDP 11 and for "normal" insertions on the CDC 6700]

CALL LVDSIN [for "destructive" insertions, CDC 6700 versions only]

CALL LVNDIN [for "nondestructive" insertions, CDC 6700 versions only]

Input Parameters:

(In COMMON/LVARGS/)

IFUNC Link of the triple; must be a random number as defined by LVGRN.

IARG Source node of the triple; must be a random number as defined by LVGRN.

NVAL Number of values (up to ten) to be inserted (default is 1).

IVALS(10) Array containing the values or sink nodes to be inserted. IVALS(i) may contain any of the following types of values:

- o Random number, as defined by LVGRN
- o Integer data; see the section on Integer Data for limitations
- o Hollerith data; see the section on Hollerith Data for limitations

ITYPE1(10) Type of each value in IVALS(i) to be inserted:

= 0 Random number (default value)

= 1 Integer data

= 2 Continuing Hollerith data

= 3 The only or final cell of a Hollerith data string

INDXON* Type of insertion to be made:

= 0 Normal insertion; the triple is always placed at the end of the (null) list. This is the default value.

= 1 Destructive insertion; the contents of the IPOSth member of the ITYP type (counting from the top or bottom of the list, depending on the sign of IPOS) are replaced by the contents of IVALS(1).

= 2 Nondestructive insertion; the contents of IVALS(1) are wedged into the list, making the new value the IPOSth member of the ITYPth type from the top or bottom of the list (depending on the sign of IPOS).

The following two variables are needed only if the operation is an indexed insertion.

IPOS LVNSRT will place the value to be inserted IPOS locations (as modified by ITYP) from the beginning or, if negative, the end of the list.

ITYP Type of value to be counted when attempting to insert a value at IPOS locations from the beginning or end of a list.

Output Parameters:

(In COMMON/LVARGS/)

IPOS Set internally to 1 (default value)

ITYP Set internally to 3 (default value)**

IVAL Set internally to IVALS(1)

NVAL Set internally to 1 (default value)

ITESTR = -4 Unsuccessful insertion attempt into full buffer made; program is terminated.

= -3 Buffer filled up before NVAL (>1) values could be inserted.

= -2 Buffer filled by this insertion; no more triples may be added.

= -1 Function did not exist prior to this insertion.

= 1 Function did exist prior to this insertion.

*PDP-11 implementation only. Entry points LVDSIN and LVNDIN are used with the CDC implementations.

**See description of Subroutine LVFNV for explanation and list of value types.

Comments:

If LVNSRT is to be used, LVFIND must first be called. The arguments to LVFIND (IFUNC and IARG) must have been defined previously by LVGRN. LVNSRT will then either create an SVL, if the list did not previously exist, or add the value to the end of an already existing function. If the function is not already in existence, a "failure" switch is set (ITESTR = -1). The following code illustrates how to add valuei, which happens to be a random number, to the "NODE1" - "LINK1" function (Note that if valuei were not a random number, an additional variable ITYP(1), would have to be set before the call to LVNSRT could be made):

```
IFUNC = LINK1
IARG = NODE1
Call LVFIND
IVALS(1) = valuei
Call LVNSRT
```

Thus far, the discussion of Subroutine LVNSRT has been of general value. The following information pertains only to indexed (destructive or nondestructive) insertions. With a destructive insertion, the contents of the IPOSth value on a list are replaced with the contents of IVALS(1). With a nondestructive insertion, the contents of IVALS(1) are wedged into a list such that the new value is at the IPOSth position.

Indexed insertions will fail under certain conditions. For example, an attempt to place a value at the fifth position in a list of length three will fail. Indexed insertions will succeed under the following rule:

Given n values of type K on a list where n can = 0,
indexed insertions will succeed for $1 \leq IPOS \leq n+1$

If indexed insertion is to be used, IPOS must be set after LVFIND has been called and LVFNV must then be called. To be safe, the formal parameter to LVFNV should be set to 0 immediately prior to the call to LVFNV. For the PDP-11 implementation, INDXON must then be set; for either of the

CDC 6700 versions, the entry point LVNDIN (for nondestructive insertion) or LVDSIN (for destructive insertion) must be called. This different procedure is necessitated by the differences in the CDC 6700 and PDP-11 FORTRAN compilers.

The following code illustrates the replacement of the third item from the bottom of the list with valuex, an integer:

```
     IFUNC = LINK1
     IARG = NODE1
     Call LVFIND
     IPOS = -3
     DUMMY = 0
     Call LVFNV (DUMMY)
     IVALS(1) = valuex
     ITYP(1) = 1

[PDP-11 Implementation]:
     INDXON = 1
     Call LVNSRT

[CDC 6700 Implementation]:
     CALL LVDSIN
```

Abstract:

LVNSRT must first determine whether or not there is room in the buffer for the new triple. If not, a failure exit is taken. A test is then made to determine whether the function already exists. If not, and if the computed location for the function is available (FL1MSK = 0 at that location), the available space pointers that follow and precede that location will be updated to point around it, and the four fields (NODSPC, LSTSPC, LNKSPC and FLGSPC) of that location will then be filled with the information appropriate to describe the triple.

If the computed location is already filled with an item that is not the head of a conflict list,* that item (a member either of an MVL or a conflict list) will be moved out of the computed location and into another determined by REGASP. The pointers in AS are updated accordingly. If the triple already occupying the computed location is the head of a conflict list, the new function will be added to that conflict list. A space is

---

*A function is the head of a conlict list if its computed address is the same as its location within the buffer.

removed from AS at the direction of REGASP, and the pointers in AS are updated accordingly via a call to LVUPDT.

If the function already exists in the buffer, it is merely a matter of adding a value to the end of an MVL, or, if the function is an SVL, of converting it to a two-valued MVL.

Destructive Insertion: First, it must be determined whether or not the IPOSth value of the requested type exists. If so, and if the function is an SVL, a value replacement occurs in LSTSPC; if the function is an MVL, the replacement occurs in NODSPC. The location (LOC) of the old value has been determined by LVFIND or LVFNV.

If the IPOSth value does not exist, LVFIND and LVFNV are called to determine whether the ($|IPOS|-1)^{th}$ value exists. If it does not, a failure return occurs. If it does, the new value is placed at the end of the list (if IPOS is positive) or in front of the first value (if IPOS is negative).

Nondestructive Insertion: The logic for nondestructive insertions parallels that of destructive insertions.

Equivalent GIRL Code:

Assume that NODE1 is the source node and LINK1 is the LINK:

1) Add random number valuei to the (null) list

```
G     NODE1   LINK1   valuei
```

2) Add integer I to the end of the list

```
G     NODE1   LINK1   "I"
```

3) Place valuei in the third location from the bottom of the list.

```
G     NODE1   LINK1   .-3  valuei
```

4) Replace the second integer value from the top of the list with the integer 10.

```
G     NODE1   LINK1-..2    "10"
```

Program Length:

CDC 6700		PDP-11
Unpacked	Packed	
605_8 (389)	1331_8 (729)	3206_8 (1670)

Subroutine Called:

LVFIND

LVFNV

LVUPDT

LVLFSH (Packed version only)

LVRTSH (Packed version only)

Called by the Following Subroutine:

LVCMPN

DELETION

Subroutine LVDLET

Function:

Deletes an entire function or the IPOSth value of the ITYPth type, counting from the top or bottom (depending on the sign of IPOS) of a list of the requested function.

Calling Format:

CALL LVDLET [for all types of deletions on the PDP-11 and for deletion of entire functions on the CDC 6700 versions]

CALL LVDLTI [for "Indexed deletions," CDC 6700 versions only]

Input Parameters:

(In COMMON/LVARGS/)

IFUNC Link of the triple; must be a random number as defined by LVGRN.

IARG Source node of the triple; must be a random number as defined by LVGRN.

IPOS Position in the MVL of the value to be deleted (number of locations from the top, if positive, and from the bottom, if negative). If ITYP is specified, only that type of value is counted in determining the position in the list. IPOS is used only for indexed deletion and must be set prior to the call to LVFNV.

ITYP Type of value to be deleted from a multivalued list (used only for indexed deletion and must be set prior to the call to LVFNV):

= 0 Random number

= 1 Integer data

= 2 Hollerith data

= 3 No specified type (default value)

ITESTR Output from LVFIND and LVFNV; indicates whether or not the IPOSth value of the function exists.

INDXON (PDP-11 version only)

= 0 Delete entire function (default)

= 1 Delete specific value as described by IPOS and ITYP.

(In COMMON/LVARGS/)

(See the description of Subroutine LVFIND for a discussion of labeled COMMON/LVADDR/).

Output Parameters:

(In COMMON/LVARGS/)

IVAL — Deleted value. If the entire list is deleted, IVAL returns the first value of the list.

ITESTR — Function indicator. If the function or specified value of that function does not exist, the attempted deletion is considered to have failed. ITESTR is actually set in LVFIND and LVFNV.

= 1 Function exists

= -1 Function does not exist

Comments:

To delete an entire function, set IARG and IFUNC prior to calling LVDLET. The first value of the function will be returned in IVAL.

If indexed deletion is to be performed, calls must be made first to LVFIND and LVFNV, with values in IARG, IFUNC, IPOS and, if necessary, ITYP. Then, for the PDP-11 version, INDXON must be set to 1 before LVDLET is called. For the CDC 6700 versions, LVDLET is entered at entry point LVDLTI. The deleted value is returned in IVAL.

Abstract:

Delete Entire Function:

LVFIND is called to determine whether the function exists. If it does not, a failure return is taken. If it does, is the function an SVL or an MVL? If it is an MVL, all locations but that at the head of the list are immediately returned to AS. The location remaining is then treated as though it were an SVL.

If the function is an SVL, the first test must determine whether or not it heads a conflict list. If it is not, the conflict list will be reconnected around the function and the vacated location will be returned to AS. If it is the head of a conflict list, the first action must be to move the next function (if any) on the list into the "head of the conflict list" location. If the moved function is also the head of an MVL, its pointers must be updated. The vacated location is then returned to AS.

Indexed Deletion:

Both LVFIND and LVFNV must be called before an indexed delete can be performed. LVFNV will return the location (LOC) of the item to be deleted. The existence of the function is tested first. If the function does not exist, a return is made. If it does exist, the SVL or MVL test follows. If it is an SVL, IPOS must = ± 1 with the proper ITYP, or the indexed delete will fail. If the function to be deleted is an MVL, indexed delete will encounter one of four situations: the value to be deleted will be in the first position, in the middle position, or in the last position in a list, or the list will be reduced to an SVL. In the fourth case, two locations are returned to AS. Each case involves a slightly different way of reconnecting the pointers within the list.

Equivalent GIRL Code:

Assume NODE1 is the source node and LINK1 is the link.

Example 1. Delete the entire function:

```
G     NODE1-LINK1
```

Example 2. Delete the I^{th} value on an MVL

```
G     NODE1+LINK1-.I
```

Program Length:

CDC 6700		PDP-11
Unpacked	Packed	
137_8 (95)	407_8 (263)	730_8 (472)

Subroutines Called:

LVFIND

LVRTSH (packed version only)

LVLFSH (packed version only)

DISK STORAGE AND RETRIEVAL OF A GRAPH

After a graph has been created, it may be conveniently stored on disk and later retrieved from disk via the Subroutines LVDUMP and LVFECH, using binary reads and writes. Although this task can be performed without these routines, their use insures that all pertinent variables will be properly defined. LVDUMP enables the user to have an entire graph, or just a part of that graph, generated in BCD format for debugging purposes.

Another advantage of this arrangement is that it makes it easy for the user to restart a program using new data. The original graph will always be retrieved whenever a new call to LVFECH is made. The user is responsible, however, for saving (and later retrieving) the values of any identifiers defined during the course of a run and must make sure that the logical unit used to store the graph is defined at the beginning of the program. This is accomplished on the CDC 6700 computer as part of the PROGRAM card information, i.e., PROGRAM TEST (...,TAPE22,..). For the PDP-11 series, a call to the RT-11 system subroutine ASSIGN must be made. For example, to store a graph on Logical Unit 10, the following statement entered just preceding the first call to LVDUMP might be used:

```
CALL ASSIGN (10, 'SY:GRAPH1.ISZ', 13, 'NEW')
```

Subroutine LVDUMP

Function:

Writes pertinent GIRS system variables and all or a part of the buffer containing the graph onto a designated local file in either binary or BCD format.

Calling Format:

CALL LVDUMP(I,J,N)

Input Parameters:

(In COMMON/LVVTR1/)

MEMSZE Length of the GIRS buffer. It must be defined at the beginning of the program.

(In COMMON/LVVSEQ/)

ISEQSZ Length of SEQSPC. It must be defined at the beginning of the program.

(Formal Parameter Set)

I Lower boundary of the portion of the GIRS buffer to be written out (BCD format).

J Upper boundary of the portion of the GIRS buffer to be written out (BCD format); 0 indicates a binary write.

N Logical unit on which the graph and variables are written.

Comments:

I and J indicate the buffer boundaries to be written out in BCD. To write out the entire buffer in BCD, set I = 1 and J = MEMSZE. For a binary write of the entire buffer, set J = 0. The logical unit onto which the graph will be written must have been previously defined.

Program Length:

CDC 6700		PDP-11
Unpacked	Packed	
252_8 (170)	241_8 (161)	605_8 (389)

Subroutines Called:

LVRTSH (packed version only)

Subroutine LVFECH

Function:

Reads (in binary format) pertinent GIRS system variables and the GIRS buffer. The latter contains a graph structure stored previously by Subroutine LVDUMP.

Calling Format:

Call LVFECH(N)

Input Parameters:

N Logical unit number as defined by the local file name containing the stored graph.

Comments:

This routine must be the first GIRS subroutine called if the program is to operate on a previously created graph. To assure that the format is correct, the graph should have been written out by the GIRS subroutine LVDUMP. The logical unit number must have been previously defined.

Program Length:

CDC 6700		PDP-11
Unpacked	Packed	
117_8 (79)	64_8 (52)	235_8 (157)

COMPRESSION/EXPANSION OF GIRS BUFFER SPACE

After a graph has been created and stored, there may still be some unused AS in the buffer. Although a buffer which is less than full results in fewer conflict lists and, therefore, a lower average data access time, the AS is, in a sense, wasted, and the user may wish to compress the buffer size to just that amount needed. When the buffer is already full and new relationships are to be added to a previously created graph (and the graph cannot be re-created economically by rerunning the original program at a larger buffer size), GIRS enables the user to expand the buffer size enough to accept new triples. Subroutine LVCMPN is designed for compressing or expanding the buffer as the situation requires.

Subroutine LVCMPN

Function:

Converts the GIRS buffer, which holds the graph, to a new size.

Calling Format:

Call LVCMPN

Input Parameters:

(In COMMON/LVVTR1/)

MEMSZE New buffer size

(In COMMON/LVTABL/)

MAPSZE Old or new GIRS buffer size, whichever is larger.

IEXTRA Quantity of free space to be added to a "minimized" graph.

MAP (mapsze) MAP must be dimensioned to MAPSZE in the user's main program.

(In COMMON/LVVTR5/) - Packed Version

BINFIL Logical unit number containing the graph in the old buffer size.

KOMPAN Compress/expand switch:

= 1 Change GIRS buffer size to MEMSZE

= 2 Compress GIRS buffer to a minimum size plus IEXTRA free spaces.

WORKSP Array which holds old graph; must be dimensioned in the user's main program to the buffer size of the old graph.

The following labeled COMMON blocks are needed for the PDP-11 and unpacked versions (all arrays in these blocks must be properly dimensioned in the user's main program):

(In COMMON/LVVTR5/)

BINFIL See previous definition.

KOMPAN See previous definition.

NODESP Dimension to old GIRS buffer size.

(In COMMON/LVVTR6/)

LISTSP Dimension to old GIRS buffer size.

(In COMMON/LVVTR7/)

LINKSP Dimension to old GIRS buffer size.

(In COMMON/LVVTR8/)

FLAGSP Dimension to old GIRS buffer size.

Output Parameters:

(In COMMON/LVVTR1/)

MEMSZE Length of the new buffer. If KOMPAN = 3, the value in MEMSZE is the totally compressed buffer size.

Comments:

This routine is not directly user callable. It can be initiated only by calling LVSETP and properly setting all input parameters for both routines. LVCMPN assumes that the graph was originally written out to disk by Subroutine LVDUMP. Also, all nodes and links saved must be given new random numbers, since the random number sequence is a function of MEMSZE and the value of each identifier must be unique. The values of the nodes and links are converted by setting identi = MAP(identi) for each identifier. This is done after LVSETP has been called, of course. For further details, refer to the Compression/Expansion Run steps, p. 21.

Subroutine Description:

The activity of Subroutine LVCMPN is divided into two parts. In the first part, the old graph is read into memory and the new buffer size (MEMSZE) is either determined or verified to be large enough to hold the graph. In the second part, a table (MAP) is set up to transform the old random numbers into new ones and the graph is then re-created.

If the compress/expand switch (KOMPAN) is $\neq 0$, LVCMPN will be called at the beginning and at the end of LVSETP. The first part of LVCMPN is executed during the first call and the second part during the second call. When LVCMPN is called for the first time, the old graph is read into memory in the same format used by both LVDUMP and LVFECH. LVFECH cannot be used, of course, since WRKSPC must be kept clear for the created graph.

LVCMPN must then determine the minimum amount of buffer space needed to store the graph, even if "minimize" is not requested, since the newly requested buffer size may not be large enough to hold the graph. Therefore, two criteria must be met:

i. The new buffer must be large enough to hold all the triples.

ii. The new buffer must be large enough to accommodate definition of all of the existing nodes and links.

To check the first criterion, the old buffer is searched and a counter (LSUMA) incremented for every location not in available space [FLlMSK = 1 in FLAGSP(i)]. The second criterion is checked by determining the total number of nodes and links in the graph, searching through the old buffer for each location which stores the head of a conflict list. As each new head of a conflict list is found, the contents of "ADDRESS" (ADDRESS has been described earlier in the section, "Computation of a Buffer Address for a Function") are updated to that location. Each conflict list is then traversed. The link is picked up, the source node is retrieved, and locations in MAP which correspond to the values of the link and source nodes are then flagged. Sink nodes which are random numbers must also be counted. To do this, it must first be determined whether the function is an SVL or MVL. If it is an SVL, and the sink node (which is stored in LISTSP for SVL's) is a random number, MAP (LISTSP (loc)) will be flagged; if it is an MVL, that MVL will be traversed, while testing for data type. MAP (NODSPC(loc)) is flagged when applicable. The total number of nodes and links can then be established by summing (in LSUMB) all the flagged locations in MAP. The minimum buffer size will then be the larger of the two summations, LSUMA and LSUMB. If an inadequate buffer size has been specified, a warning message will be printed out stating the minimum allowable buffer size, and the program will stop. If the program has been requested to "minimize" the buffer, KPRIME (from COMMON/LVRAND/) will be computed (as described in the description of Subroutine LVSETP) before control is returned to LVSETP. LVSETP will then initialize WRKSPC and once again call upon LVCMPN, this time to perform the second part of its function.

The second part of LVCMPN is itself made up of two separate tasks: (a) setting up the old-to-new random number correspondence table (MAP), and (b) re-creating the graph in the new buffer. Flags are set at each location in MAP that corresponds to a node or a link value. That is, if a value of one of the nodes or links is L, then MAP(L) is set to 1. To set up the correspondence table, LVCMPN searches MAP for flagged locations and gives each flagged location a random number returned from Subroutine LVGRN;

the MAP array is given values such that a random number from the old graph is given a different random number for the new graph. This is necessary because the sequence of random numbers changes for different buffer sizes.

The second task performed by LVCMPN during its second calling is to re-create the graph in the new buffer. This task involves searching the old graph for all the triples, converting all old random numbers to new ones, and then re-inserting the triples into the new graph. The algorithm is similar to that used in the first part of LVCMPN to determine the total number of nodes and links (criterion two). The old buffer is searched for heads of conflict lists. These locations become the old computed ADDRESS. The conflict list is then traversed, extracting the link and computing the value for the source node. Old random numbers are converted to new ones as follows:

NEWNUM = MAP(OLDNUM)

The function is then examined to see whether it is an SVL or an MVL. If it is an SVL, the sink node (if a random number) is extracted and converted. If the list is multivalued, it is traversed and the sink node converted, if necessary, and inserted into the new buffer.

Program Length:

CDC 6700		PDP-11
Unpacked	Packed	
362_8 (242)	366_8 (246)	1446_8 (806)

Subroutines Called:

LVFIND

LVNSRT

LVGRN

LVRTSH (Packed version only)

LVLFSH (Packed version only)

Called by the Following Subroutine:

LVSETP

CONVERSION OF GIRS BUFFER FROM A PACKED TO AN UNPACKED VERSION, AND VICE VERSA

There are two versions of GIRS residing on the CDC 6700 computing system. In the first version, all four information fields are packed into a single array. This is called the "packed" version. In the second, each of the four fields is contained within its own array, hence the name, "unpacked" version. The reason for providing the two different versions is to accommodate the time-versus-space trade-off consideration inherent in many programming decisions. Since program requirements may change from time to time, the ability to convert from one version to the other is useful. The subroutines LVPACK and LVUNPK are used for this purpose.

Subroutine LVUNPK

Function:

Converts a packed GIRS buffer into an unpacked GIRS buffer, i.e., converts a GIRS buffer, in which all four of the information fields are contained in one array, into one in which each information field is in a separate array.

Calling Format:

CALL LVUNPK(L)

Input Parameters:

(Formal Parameter Set)

L Logical unit number as defined by the local file name.

Comments:

This routine is called directly by the user. Also, in order to be in the proper format, the graph should have been written out by GIRS subroutine LVDUMP (packed version).

Program Length:

CDC 6700

Unpacked Version Only

107_8 (71)

Subroutine Called:

LVRTSH

Subroutine LVPACK

Function:

Converts an unpacked GIRS buffer to a packed GIRS buffer; that is, it packes the information field from each of four arrays NODSPC, LSTSPC, LNKSPC, and FLGSPC (unpacked version) into four information fields in one array (WRKSPC).

Calling Format:

Call LVPACK (NODSPC, LSTSPC, LNKSPC, L)

Input Parameters:

(Formal Parameter Set)

NODSPC (memsze)
LSTSPC (memsze)
LNKSPC (memsze) — These arrays must be properly dimensioned to MEMSZE in the calling routine.

L — Logical unit number of the local file name which contains the old GIRS buffer.

(In COMMON/LVVSEQ/)

ISEQSZ — Length of SEQSPC (Sequence space); must be defined when program is initialized.

Comments:

This routine is called directly by the user. Also, to assure proper format, the graph should have been written out by the GIRS subroutine LVDUMP (unpacked version).

Program Length:

CDC 6700

Packed Version Only

222_8 (146)

Subroutines Called:

LVRTSH

LVLFSH

INTERNAL ROUTINES

The following routines are not user callable.

Subroutine LVUPDT

Function:

Updates the Available Space (AS) pointers in WRKSPC and the REGister of Available SPace (REGASP) to prepare for the insertion of a triple.

Calling Format:

Call LVUPDT

Input Parameters:

(In COMMON/LVVTR1/)

REGASP Points to the AS location in the buffer to be used next to store either a function which is not the head of conflict list or a value on an MVL.

NODSPC* AS up-pointer.

(In COMMON/LVVTR2/)**

LSTSPC* AS down-pointer.

Output Parameters:

Same as input parameters

Subroutine Description:

When a value is to be added to an MVL, or a function is to be added to a conflict list, a cell must be removed from AS. The AS pointer in NODSPC which points to the location to be removed must be changed to point to the location just ahead of the one to be removed. The AS pointer in LSTSPC which points to the location to be removed must be changed to point to the location in AS just behind the one to be removed. Before control is returned, REGASP is given a new location in AS.

Program Length:

CDC 6700		PDP-11
Unpacked	Packed	
7_8 (7)	34_8 (28)	45_8 (37)

*WRKSPC in the packed version contains all four information fields.

**Not in packed version.

Subroutine Called:

LVRTSH (Packed version only)

Called by the Following Subroutine:

LVNSRT

Function LVRTSH

Function:

Performs a noncircular, zero-filled right shift.

Calling Format:

NEWORD = LVRTSH(IWRD,IBITS)

Input Parameters:

(Formal Parameter Set)

IWRD Contents to be shifted.

IBITS Number of bits to the right involved in the shift.

Program Length:

CDC 6700
20_8 (16)

Called by the Following Subroutines:

LVFNV

LVUNPK (unpacked version only)

LVDLET (packed version only)

LVFIND (packed version only)

LVPACK (packed version only)

LVNSRT (packed version only)

LVUPDT (packed version only)

LVDUMP (packed version only)

LVCMPN (packed version only)

Function LVLFSH

Function:

Performs a noncircular, zero-filled left shift.

Calling Format:

NEWORD = LVLFSH(IWRD,IBITS)

Input Parameters:

(Formal Parameter Set)

IWRD Contents to be shifted.

IBITS Number of bits involved in the shift.

Program Length:

CDC 6700
31_8 (25)

Called by the Following Subroutines:

LVFNV

LVSETP (packed version only)

LVDLET (packed version only)

LVPACK (packed version only)

LVNSRT (packed version only)

DECK SETUPS AND COMMAND SEQUENCES

GENERAL DISCUSSION

GIRS may be used directly via user calls to the GIRS subroutines or indirectly with the GIRL language. In either case, for all implementations of GIRS, the object code for the driving program must precede the object code for the GIRS routines in any LINK-LOAD.

It is generally more advantageous for the user to use GIRS indirectly via GIRL, since GIRL not only includes all the capabilities of GIRS but also spares the user from concern over setting up all the labeled commons and initializing pertinent variables. The command sequences and FORTRAN statements needed to preprocess, compile, link, and execute GIRL/GIRS programs on the CDC 6700 and the PDP-11 follow. The deck setups for batch-entered GIRL/GIRS programs on the CDC 6700 have been excerpted from Berkowitz.[2]

INDIRECT USE OF GIRS SUBROUTINES VIA GIRL

CDC 6700:

(Unpacked Version)	Card No.
job card	(1)
charge card	(2)
ATTACH, PREP, PREPBIN, ID=CAIZ.	(3)
ATTACH, GIRS, GIRSBIN, ID=CAIZ.	(4)
ATTACH, TAPE99, GIRLGRAPH, ID=CAIZ.	(5)
LOAD, PREP.	(6)
GIRS.	(7)
FTN, I=TAPE8.	(8)
FTN. (used only if purely FORTRAN routines are to be run)	(9)
LOAD, LGO.	(10)
GIRS.	(11)
end of record	(12)
memsze option1 option2 ...	(13)
PROGRAM name	(14a)
or	
$ SUBROUTINE name	(14b)
non-DATA specification statements	.
G DEFINE string (optional)	(15)
DATA string (optional)	(16)
G EXECUTE	(17)
GIRL/FORTRAN executable code (no END statement)	.
G COMPLETE	(18)

```
   other GIRL/FORTRAN routines                                    .
/  COMPLETE                                                       (19)
end of record                                                     (20)
   purely FORTRAN routines                                        .
end of record                                                     (21)
   data                                                           .
end of record                                                     (22)
end of file                                                       (23)
```

(Packed Version)

Insert the following cards after Card (7):

```
UNLOAD, GIRS, PREP, TAPE99.                                       (7a)
ATTACH, GIRS, GIRSPACKBIN, ID=CAIZ.                               (7b)
```

Notes:

(1) End-of-record is accomplished by a simultaneous 7/8/9 punch in Column 1. End-of-file is accomplished by a simultaneous 6/7/8/9 punch in Column 1.

(2) In the GIRL/FORTRAN program, GIRL statements are declared by placing a G in Column 1. Continuation cards are handled as in FORTRAN.

(3) The option Card (13)* has the following entries:

memsize — An integer of at most six digits that stipulates the buffer size and the number of possible nodes that the graph may contain. There is no default; some integer must be entered, right justified, into the first six columns.

*XX — An integer of at most two digits preceded by an asterisk (*) declares the file number on which an old graph is stored. Default implies that a new graph is to be set up. The file which contains the old graph should be attached by means of a control card:

```
ATTACH, TAPEXX, pfname                                            (6a)
```

where XX is the file number and pfname is a user permanent file name.

---

*Except for the first entry (first six columns), the other entries are optional and may appear in any order, separated by at least one space or comma.

$IIIIII	An integer of at most six digits preceded by a dollar sign ($) declares the size of SEQ. Default size is one location.
PACK	Sets up code for packed version of GIRL. Default is unpacked version.
PRINT	Prints GIRL program on output file. Default is no-print.
COMMENTS	Places GIRL code with a G in Column 1 into preprocessed FORTRAN code. Default is no-comment.
LXX	An integer of at most two digits preceded by a letter L declares the maximum allowable levels of paranthesization.
NOSAVE	Eliminates the saved-index facility, and is therefore appropriate for short multivalue lists. (See the discussion of "saved index" in the previous section).

Setup For Cataloging a Graph (Prior to Compression or Expansion):

Compression or expansion of graph memory leads to a reordering of node addresses relative to the cell address at which node-link-node triples are stored. Therefore, one must save node addresses (which are of special interest to programs that manipulate the graph) so that the compress/expand program (described in the next subsection) can retrieve the node address mappings. In the following setup, these addresses are represented by varl, ..., varn. The metasymbol pfname refers to a permanent file name to be assigned by the user.

The deck setup is the same as for any GIRL/FORTRAN program with the following additions:

Card (4) is followed by
REQUEST, TAPE17, *PF.

Card (11) is followed by
CATALOG, TAPE17, pfname

Card (13) should include the option PRINT.

Card (14a) should have (TAPE17,...) followed by the program name, where the dots indicate other files used by the program.

Card (18) should be preceded by
CALL LVDUMP(0,0,17)

WRITE(17) n, varl,...,varn

Setup For Previously Created and Stored Graph (And Compression or Expansion):

The deck setup is the same as for any GIRL/FORTRAN run with the following additions:

Card (4) is followed by
ATTACH, TAPE88, pfname.
REQUEST,TAPE27, *PF.

Card (11) is followed by
CATALOG, TAPE27, pfname.

Card (13) should include either a declaration of the size of the new graph or a request to minimize the graph to the smallest possible size with an option of adding some free space. The forms for this option are as follows: /I where I is an integer value; /M; or /M+I where I is an integer value. The card should also include .*88.

Card (14a) should have (TAPE88, TAPE27) following the PROGRAM name.

The deck is completed as follows:

```
     COMMON /LSAVE/ n, varl,..., varn
G    EXECUTE
     READ(88) n, varl,..., varn
          .
          .
          .
     CALL LVDUMP(0,0,27)
     WRITE(27) n, varl,..., varn
G    COMPLETE
/    COMPLETE
     end of record
     end of file
```

PDP-11:

Assume that all the files are to reside on the system disk* and that the GIRL program USER.GRL is to be preprocessed and executed. The preprocessor accepts the GIRL, FORTRAN, and list file names in Command String

---

*The graph used by the preprocessor 'PRPGRF.BIN' must reside on the system disk drive ('SY:').

Interpreter format with default file extension names respectively: GRL, FOR, and LST. The list file is optional. The card images (13) and (14b) through (19) from the Batch-Entry Deck Setup for the CDC 6700 are included in USER.GRL. The preprocessor will create a FORTRAN file and (as an option) a GIRL listing. A copy of the GIRL listing will also be sent to the terminal if the PRINT option has been requested. These files are to be named USER.FOR and USER.LST, respectively. The periods and asterisks at the beginning of lines are system prompt characters. The terminal dialog involved in preprocessing, compiling, linking, and executing the GIRL program USER.GRL is as follows:

```
                                                        Line No.
.R PREP                                                   (1)
ALL REAL VARIABLES MUST BE DECLARED
ERRORS ARE FLAGGED BY ****ERROR
PLEASE ENTER FILE NAMES IN COMMAND STRING FORM
*USER=USER                                                (2)
  or, if a list file is also desired:
*USER,USER=USER                                           (2a)
.R FORTRAN                                                (3)
*USER=USER/W                                              (4)
*^C  (control C)                                          (5)
.R LINK                                                   (6)
*USER=USER,GIRS/F                                         (7)
*^C                                                       (8)
.R USER                                                   (9)
```

Please note that, for GIRL programs using old graphs stored on disk, files containing those graphs must be defined via a Call to Subroutine ASSIGN. This statement must be included in the main program <u>immediately</u> following the card:

```
G     EXECUTE
```

DIRECT USE OF GIRS SUBROUTINES

CDC 6700:

Delete card number (3) and card numbers (5) through (7) of the deck setup of page 63 and add the following:

```
ATTACH, TAPE8, userdriving program.
```

PDP-11:

Eliminate lines (1) and (2) of the dialogue given for using GIRS indirectly (page 67), and insert the following statements within the main routine in the user's FORTRAN program:

(To create a new graph):

```
      CALL ASSIGN(27,'RKn:USER.GRF',12,'NEW')
           .
           .
           .
      CALL LVDUMP(0,0,27)
C     SAVE NODE AND LINK VALUES ON LUN 27
```

(To use a previously created graph):

```
      CALL ASSIGN(27,'RKn:USER.GRF',12,'OLD')
      CALL LVFECH(27)
C     DO NOT CALL LVSETP SINCE THAT WILL WIPE OUT THE GRAPH JUST
      READ IN.
C     READ IN SAVED NODE AND LINK VALUES.
```

These instructions are to be inserted at the beginning of the program.

PROPOSED EXTENSIONS TO GIRS

The following extensions to GIRS are being considered:

o Creation of a paged or out-core version of GIRS. This idea is described in Berkowitz.[1]

o Creation of a program to interactively access and modify the graph.

o Creation of a sequential space (SEQSPC) within GIRS. Such a space would facilitate rapid storage and retrieval of long Hollerith strings by setting up a static, linear storage area with pointers to the beginning of each block being stored within the graph. An example of the use of sequential space is given in Berkowitz.[2]

o Use of a list-naming procedure, such as that described in Berkowitz.[3]

o Recovery of temporarily created random nodes.

ACKNOWLEDGMENTS

The GIRS and GIRL software described was designed by Dr. S. Berkowitz. The initial coding of the basic GIRS/GIRL routines was carried out by Ann E. Bandurski.

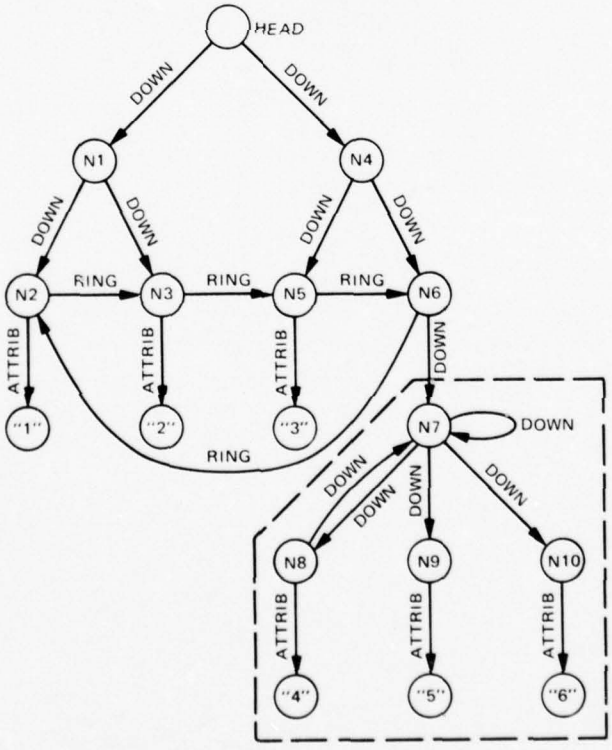
APPENDIX A

SAMPLE PROGRAMS

Sample programs have been provided to illustrate the effectiveness of GIRS/GIRL in handling pointer manipulation problems. Three programs are offered, each coded first in GIRL/FORTRAN and then in all FORTRAN (as generated by the GIRL preprocessor). All FORTRAN versions of the programs call the GIRS routines directly. These programs are coded for the PDP-11 and follow the program descriptions.

Program (1):

Create a graph having the relationships shown in the following diagram with "HEAD" as starting (top) node and with nodes N1, ..., N10 just underneath, of which nodes N2, N3, N5, N8, N9 and N10 are linked to "terminal" nodes that contain the attribute values "1", "2", ..., "6", respectively. Nodes at the second level shall be connected via "RING" links. The "DOWN" links are used to describe the directions of the relationships. Store this graph onto disk.



Program (2):

Read the previously created graph from disk and modify it by deleting that part of the structure enclosed by the dashed line in the illustration.

Program (3):

Simulate a computer resource allocator (operating system) with the following resources:

a) Memory three blocks of 50 words
one block of 100 words

b) Two tape drives

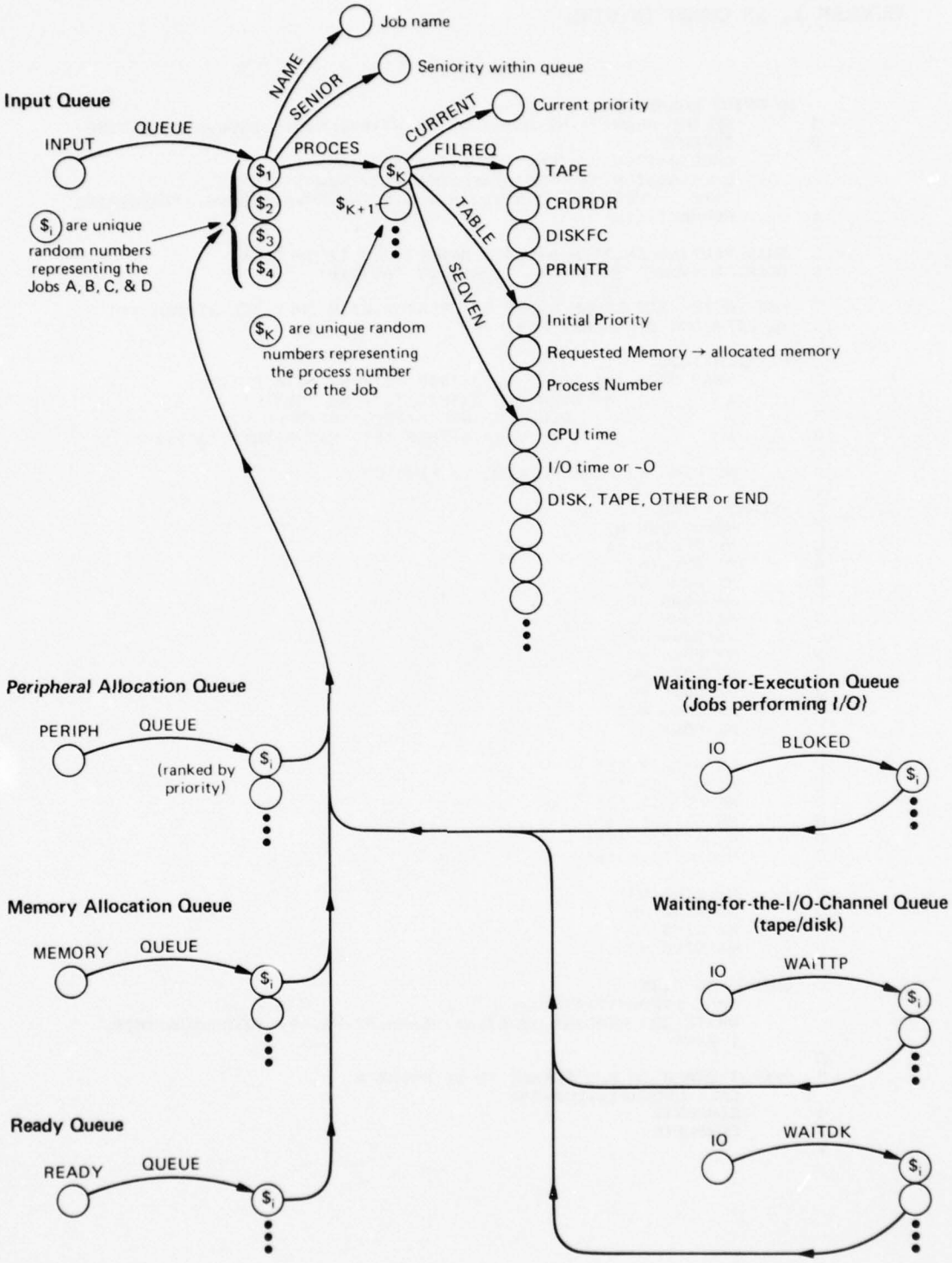
c) Two printers

d) Two card readers

e) Six disk files

f) One I/O channel

Resource allocations must be made by program priority as modified by seniority within any particular waiting queue. The following diagram shows the relationships involved in Program 3.


Job name
Seniority within queue
Current priority
NAME
SENIOR
CURRENT
Input Queue
INPUT
QUEUE
PROCES
FILREQ
TAPE
CRDRDR
DISKFC
PRINTR
TABLE
SEQVEN
Initial Priority
Requested Memory → allocated memory
Process Number
CPU time
I/O time or -O
DISK, TAPE, OTHER or END
$i are unique random numbers representing the Jobs A, B, C, & D
$K are unique random numbers representing the process number of the Job
Peripheral Allocation Queue
PERIPH
QUEUE
(ranked by priority)
Waiting-for-Execution Queue
(Jobs performing I/O)
IO
BLOKED
Memory Allocation Queue
MEMORY
QUEUE
Waiting-for-the-I/O-Channel Queue
(tape/disk)
IO
WAITTP
IO
WAITDK
Ready Queue
READY
QUEUE

PROGRAM 1, AS CODED IN GIRL

```
  50 PRINT COMMENTS
G       DEFINE HEAD,N1,N2,N3,N4,N5,N6,N7,N8,N9,N10,DOWN,ATTRIB,RING
G       EXECUTE
        CALL ASSIGN(22,'RK0:GRAPH.BIN',13,'NEW')
        CALL ASSIGN(24,'RK0:GRAPH.BCD',13,'NEW')
        TYPE 1, HEAD,N1,N2,N3,N4,N5,N6,N7,N8,N9,N10,DOWN,ATTRIB,RING
1       FORMAT(11I6)
C
C  THIS ROUTINE CREATES A GRAPH AND STORES IT ON DISK.
C  TERMINAL NODES HAVE DATA LINKED BY "ATTRIB"
C
C  THE ENTIRE STRUCTURE COULD BE CREATED WITH ONLY TWO STATEMENTS
C  OR WITH ONE STATEMENT PER LINK
C
C  FIRST METHOD:
G       HEAD DOWN (N1 DOWN (N2 ATTRIB "1", N3 ATTRIB "2"),
G       1          N4 DOWN (N5 ATTRIB "3", N6 DOWN N7
G       1              DOWN (N7,N8 (ATTRIB "4",DOWN N7),
G       1                    N9 ATTRIB "5", N10 ATTRIB "6")))
C
G       N2 RING N3 RING N5 RING N6 RING N2
C
C  SECOND METHOD
G       HEAD DOWN N1
G       HEAD DOWN N4
G       N1 DOWN N2
G       N1 DOWN N3
G       N4 DOWN N5
G       N4 DOWN N6
G       N6 DOWN N7
G       N7 DOWN N7
G       N7 DOWN N8
G       N7 DOWN N9
G       N7 DOWN N10
G       N8 DOWN N7
C
G       N2 ATTRIB "1"
G       N3 ATTRIB "2"
G       N5 ATTRIB "3"
G       N8 ATTRIB "4"
G       N9 ATTRIB "5"
G       N10 ATTRIB "6"
C
G       N2 RING N3
G       N3 RING N5
G       N5 RING N6
G       N6 RING N2
C
C  OUTPUT TO DISK
        CALL LVDUMP(0,0,22)
        WRITE(22) HEAD,N1,N2,N3,N4,N5,N6,N7,N8,N9,N10,DOWN,ATTRIB,
        1 RING
C
C  OUTPUT GRAPH IN BCD FORMAT TO BE PRINTED
        CALL LVDUMP(1,100,24)
G       COMPLETE
/       COMPLETE
*
```

PROGRAM 1, AS CODED IN FORTRAN FOR GIRS

```
      IMPLICIT INTEGER(A-Z)
      COMMON /LVARGS/ LVFUNC,LVVARG,LVVPOS,LVVTYP,LVVAL,
      1 LVVNVL,LVSKIP,LVVTR,LVVINC,LVNDXN,LVVALS(10),LVTYPE(10)
      COMMON /LVVSEQ/ LVSIZE,LVSEQ1,LVSEQ2,SEQSPC(   1)
      COMMON /LVTABL/ LVTSZE,LVEXTR,LVMAP(   1)
      COMMON /LVVTR5/ LVFILE,LVCMPR,NODESP(   1)
      1 /LVVTR6/ LISTSP(   1) /LVVTR7/ LINKSP(   1)/LVVTR8/ FLAGSP(   1)
      COMMON/LVRAND/LVKPRM,LVKS,LVKY,LVKDY,LVKDX,LVTEMP
      COMMON/LVVTR1/LVVSZE,LVVGSP,NODSPC(  50) /LVVTR2/
      1 LSTSPC(  50)/LVVTR3/LNKSPC(  50) /LVVTR4/FLGSPC(  50)
      DATA LVTYPE /0,0,0,0,0,0,0,0,0,0/
      LVVSZE=  50
      LVFILE= 0
      LVCMPR=0
      LVSIZE=    1
      LVSKIP=0
      LVKPRM= 3
      LVTSZE=   1
      LVVNVL=1
      LVVPOS=1
      LVVTYP=3
      LVNDXN=0
      LVEXTR=   0
      CALL LVSETP
      CALL LVGRN(HEAD  )
      CALL LVGRN(N1    )
      CALL LVGRN(N2    )
      CALL LVGRN(N3    )
      CALL LVGRN(N4    )
      CALL LVGRN(N5    )
      CALL LVGRN(N6    )
      CALL LVGRN(N7    )
      CALL LVGRN(N8    )
      CALL LVGRN(N9    )
      CALL LVGRN(N10   )
      CALL LVGRN(DOWN  )
      CALL LVGRN(ATTRIB)
      CALL LVGRN(RING  )
      GO TO 25001
25000 CONTINUE
      CALL ASSIGN(22,'RK0:GRAPH.BIN',13,'NEW')
      CALL ASSIGN(24,'RK0:GRAPH.BCD',13,'NEW')
      TYPE 1, HEAD,N1,N2,N3,N4,N5,N6,N7,N8,N9,N10,DOWN,ATTRIB,RING
1     FORMAT(11I6)
C
C  THIS ROUTINE CREATES A GRAPH AND STORES IT ON DISK.
C  TERMINAL NODES HAVE DATA LINKED BY "ATTRIB"
C
C  THE ENTIRE STRUCTURE COULD BE CREATED WITH ONLY TWO STATEMENTS
C  OR WITH ONE STATEMENT PER LINK
C
C  FIRST METHOD:
C       HEAD DOWN (N1 DOWN (N2 ATTRIB "1", N3 ATTRIB "2"),
C       1          N4 DOWN (N5 ATTRIB "3", N6 DOWN N7
C       1              DOWN (N7,N8 (ATTRIB "4",DOWN N7),
C       1                   N9 ATTRIB "5", N10 ATTRIB "6")))
      LVVAL=HEAD
      LVVARG=LVVAL
      LVFUNC=DOWN
      CALL LVFIND
      LVV 1=LVVAL
      LVV 2=LVFUNC
```

```
      LVV 3=LVVARG
      LVVALS(1)=N1
      CALL LVNSRT
      LVVARG=LVVAL
      LVFUNC=DOWN
      CALL LVFIND
      LVV 4=LVVAL
      LVV 5=LVFUNC
      LVV 6=LVVARG
      LVVALS(1)=N2
      CALL LVNSRT
      LVVARG=LVVAL
      LVFUNC=ATTRIB
      CALL LVFIND
      LVVALS(1)=1
      LVTYPE(1)=1
      CALL LVNSRT
      LVFUNC=LVV 5
      LVVARG=LVV 6
      CALL LVFIND
      LVVALS(1)=N3
      CALL LVNSRT
      LVVARG=LVVAL
      LVFUNC=ATTRIB
      CALL LVFIND
      LVVALS(1)=2
      LVTYPE(1)=1
      CALL LVNSRT
      LVFUNC=LVV 2
      LVVARG=LVV 3
      CALL LVFIND
      LVVALS(1)=N4
      CALL LVNSRT
      LVVARG=LVVAL
      LVFUNC=DOWN
      CALL LVFIND
      LVV 4=LVVAL
      LVV 5=LVFUNC
      LVV 6=LVVARG
      LVVALS(1)=N5
      CALL LVNSRT
      LVVARG=LVVAL
      LVFUNC=ATTRIB
      CALL LVFIND
      LVVALS(1)=3
      LVTYPE(1)=1
      CALL LVNSRT
      LVFUNC=LVV 5
      LVVARG=LVV 6
      CALL LVFIND
      LVVALS(1)=N6
      CALL LVNSRT
      LVVARG=LVVAL
      LVFUNC=DOWN
      CALL LVFIND
      LVVALS(1)=N7
      CALL LVNSRT
      LVVARG=LVVAL
      LVFUNC=DOWN
      CALL LVFIND
      LVV 7=LVVAL
      LVV 8=LVFUNC
      LVV 9=LVVARG
      LVVALS(1)=N7
      CALL LVNSRT
      LVFUNC=LVV 8
      LVVARG=LVV 9
      CALL LVFIND
      LVVALS(1)=N8
      CALL LVNSRT
      LVVARG=LVVAL
      LVV10=LVVARG
      LVFUNC=ATTRIB
      CALL LVFIND
      LVVALS(1)=4
      LVTYPE(1)=1
      CALL LVNSRT
      LVVAL=LVV10
      LVVARG=LVVAL
      LVFUNC=DOWN
      CALL LVFIND
      LVVALS(1)=N7
      CALL LVNSRT
      LVFUNC=LVV 8
      LVVARG=LVV 9
      CALL LVFIND
      LVVALS(1)=N9
      CALL LVNSRT
      LVVARG=LVVAL
      LVFUNC=ATTRIB
      CALL LVFIND
      LVVALS(1)=5
      LVTYPE(1)=1
      CALL LVNSRT
      LVFUNC=LVV 8
      LVVARG=LVV 9
      CALL LVFIND
      LVVALS(1)=N10
      CALL LVNSRT
      LVVARG=LVVAL
      LVFUNC=ATTRIB
      CALL LVFIND
      LVVALS(1)=6
      LVTYPE(1)=1
      CALL LVNSRT
C
C     N2 RING N3 RING N5 RING N6 RING N2
      LVVAL=N2
      LVVARG=LVVAL
      LVFUNC=RING
      CALL LVFIND
      LVVALS(1)=N3
      CALL LVNSRT
      LVVARG=LVVAL
      LVFUNC=RING
      CALL LVFIND
      LVVALS(1)=N5
      CALL LVNSRT
      LVVARG=LVVAL
      LVFUNC=RING
      CALL LVFIND
      LVVALS(1)=N6
      CALL LVNSRT
      LVVARG=LVVAL
      LVFUNC=RING
      CALL LVFIND
      LVVALS(1)=N2
      CALL LVNSRT
C
C  SECOND METHOD
C     HEAD DOWN N1
      LVVAL=HEAD
```

```
      LVVARG=LVVAL
      LVFUNC=DOWN
      CALL LVFIND
      LVVALS(1)=N1
      CALL LVNSRT
C     HEAD DOWN N4
      LVVAL=HEAD
      LVVARG=LVVAL
      LVFUNC=DOWN
      CALL LVFIND
      LVVALS(1)=N4
      CALL LVNSRT
C     N1 DOWN N2
      LVVAL=N1
      LVVARG=LVVAL
      LVFUNC=DOWN
      CALL LVFIND
      LVVALS(1)=N2
      CALL LVNSRT
C     N1 DOWN N3
      LVVAL=N1
      LVVARG=LVVAL
      LVFUNC=DOWN
      CALL LVFIND
      LVVALS(1)=N3
      CALL LVNSRT
C     N4 DOWN N5
      LVVAL=N4
      LVVARG=LVVAL
      LVFUNC=DOWN
      CALL LVFIND
      LVVALS(1)=N5
      CALL LVNSRT
C     N4 DOWN N6
      LVVAL=N4
      LVVARG=LVVAL
      LVFUNC=DOWN
      CALL LVFIND
      LVVALS(1)=N6
      CALL LVNSRT
C     N6 DOWN N7
      LVVAL=N6
      LVVARG=LVVAL
      LVFUNC=DOWN
      CALL LVFIND
      LVVALS(1)=N7
      CALL LVNSRT
C     N7 DOWN N7
      LVVAL=N7
      LVVARG=LVVAL
      LVFUNC=DOWN
      CALL LVFIND
      LVVALS(1)=N7
      CALL LVNSRT
C     N7 DOWN N8
      LVVAL=N7
      LVVARG=LVVAL
      LVFUNC=DOWN
      CALL LVFIND
      LVVALS(1)=N8
      CALL LVNSRT
C     N7 DOWN N9
      LVVAL=N7
      LVVARG=LVVAL
      LVFUNC=DOWN
      CALL LVFIND
      LVVALS(1)=N9
      CALL LVNSRT
C     N7 DOWN N10
      LVVAL=N7
      LVVARG=LVVAL
      LVFUNC=DOWN
      CALL LVFIND
      LVVALS(1)=N10
      CALL LVNSRT
C     N8 DOWN N7
      LVVAL=N8
      LVVARG=LVVAL
      LVFUNC=DOWN
      CALL LVFIND
      LVVALS(1)=N7
      CALL LVNSRT
C
C     N2 ATTRIB '1'
      LVVAL=N2
      LVVARG=LVVAL
      LVFUNC=ATTRIB
      CALL LVFIND
      LVVALS(1)=1
      LVTYPE(1)=1
      CALL LVNSRT
C     N3 ATTRIB '2'
      LVVAL=N3
      LVVARG=LVVAL
      LVFUNC=ATTRIB
      CALL LVFIND
      LVVALS(1)=2
      LVTYPE(1)=1
      CALL LVNSRT
C     N5 ATTRIB '3'
      LVVAL=N5
      LVVARG=LVVAL
      LVFUNC=ATTRIB
      CALL LVFIND
      LVVALS(1)=3
      LVTYPE(1)=1
      CALL LVNSRT
C     N8 ATTRIB '4'
      LVVAL=N8
      LVVARG=LVVAL
      LVFUNC=ATTRIB
      CALL LVFIND
      LVVALS(1)=4
      LVTYPE(1)=1
      CALL LVNSRT
C     N9 ATTRIB '5'
      LVVAL=N9
      LVVARG=LVVAL
      LVFUNC=ATTRIB
      CALL LVFIND
      LVVALS(1)=5
      LVTYPE(1)=1
      CALL LVNSRT
C     N10 ATTRIB '6'
      LVVAL=N10
      LVVARG=LVVAL
      LVFUNC=ATTRIB
      CALL LVFIND
      LVVALS(1)=6
      LVTYPE(1)=1
      CALL LVNSRT
C
```

```
C       N2 RING N3
        LVVAL=N2
        LVVARG=LVVAL
        LVFUNC=RING
        CALL LVFIND
        LVVALS(1)=N3
        CALL LVNSRT
C       N3 RING N5
        LVVAL=N3
        LVVARG=LVVAL
        LVFUNC=RING
        CALL LVFIND
        LVVALS(1)=N5
        CALL LVNSRT
C       N5 RING N6
        LVVAL=N5
        LVVARG=LVVAL
        LVFUNC=RING
        CALL LVFIND
        LVVALS(1)=N6
        CALL LVNSRT
C       N6 RING N2
        LVVAL=N6
        LVVARG=LVVAL
        LVFUNC=RING
        CALL LVFIND
        LVVALS(1)=N2
        CALL LVNSRT
C
C  OUTPUT TO DISK
        CALL LVDUMP(0,0,22)
        WRITE(22) HEAD,N1,N2,N3,N4,N5,N6,N7,N8,N9,N10,DOWN,ATTRIB,
        1 RING
C
C  OUTPUT GRAPH IN BCD FORMAT TO BE PRINTED
        CALL LVDUMP(1,100,24)
        STOP
25001   CONTINUE
        GO TO 25000
        END
*
```

CONTENTS OF GIRS BUFFER AFTER PROGRAM 1 COMPLETION

GIRS MEMORY DUMP (IN OCTAL)

REGASP= 3
MEMSZE= 50 PRIME= 3 SEED= 1 NROW= 2 KDNODE= 9 TEMP= 32
SEQSIZE= 1 MAPSIZE= 1

LOC	NODSPC	LSTSPC	LNKSPC	FLGSPC	OCTAL COUNTER
1	5	40	40	140	1
2	2	6	62	140	2
3	61	14	0	0	3
4	41	10	0	0	4
5	30	3	5	115	5
6	6	13	2	140	6
7	21	62	7	114	7
8	4	53	0	0	10
9	53	16	0	0	11
10	12	20	20	140	12
11	13	21	6	140	13
12	3	31	0	0	14
13	40	50	15	114	15
14	11	24	0	0	16
15	54	25	0	0	17
16	20	26	12	140	20
17	21	62	21	354	21
18	21	1	22	354	22
19	21	62	23	114	23
20	16	33	0	0	24
21	17	34	0	0	25
22	21	12	51	354	26
23	37	37	37	140	27
24	30	2	50	111	30
25	14	41	0	0	31
26	30	4	32	115	32
27	24	54	0	0	33
28	25	44	0	0	34
29	55	45	0	0	35
30	30	5	36	115	36
31	50	50	27	140	37
32	27	22	1	140	40
33	31	4	0	0	41
34	30	1	42	115	42
35	30	6	43	115	43
36	34	55	0	0	44
37	35	56	0	0	45
38	56	57	0	0	46
39	57	61	0	0	47
40	21	27	30	354	50
41	40	12	26	110	51
42	40	20	52	114	52
43	10	11	0	0	53
44	33	17	0	0	54
45	44	35	0	0	55
46	45	46	0	0	56
47	46	47	0	0	57
48	40	37	60	114	60
49	47	3	0	0	61
50	62	2	13	140	62

PROGRAM 2, AS CODED IN GIRL

```
  50 PRINT COMMENTS *23
        COMMON /LINKS/ DOWN,ATTRIB,RING
G       EXECUTE
        CALL ASSIGN(23,'RK0:GRAPH.BIN',13,'OLD')
        CALL ASSIGN(25,'RK0:GRAPH1.BCD',14,'NEW')
        CALL ASSIGN(27,'RK0:GRAPH1.BIN',14,'NEW')
        READ(23) HEAD,N1,N2,N3,N4,N5,N6,N7,N8,N9,N10,DOWN,ATTRIB,
        1 RING
C
C  THIS ROUTINE READS IN A GRAPH AND IS A DRIVER TO DELETE THE
C  SUBSTRUCTURE BELOW N6 AND THEN SAVE NEW GRAPH
C
        CALL DELNOD(N6)
C
C  OUTPUT MODIFIED GRAPH TO DISK
        CALL LVDUMP(0,0,27)
        WRITE(27) HEAD,N1,N2,N3,N4,N5,N6,N7,N8,N9,N10,DOWN,ATTRIB,
        1 RING
C
C  OUTPUT GRAPH IN BCD FORMAT TO BE PRINTED
        CALL LVDUMP(1,50,25)
G       COMPLETE



$       SUBROUTINE DELNOD(NOD)
        COMMON /LINKS/ DOWN,ATTRIB,RING
G       DEFINE NODLST,LNKLST
G       EXECUTE
C
C  DELETE ENTIRE STRUCTURE BELOW 'NOD', ONE LEVEL AT A TIME,
C  ('BREADTH FIRST' SEARCH) THUS ELIMINATING THE POSSIBILITY
C  OF INFINITE LOOPS.
C
        NODE=NOD
 5      LINK=DOWN
C
C  PICK UP ALL POINTERS AT ONE LEVEL
 6      I=0
G10     NODE+LINK.'*I=I+1*/20'SINK
C
C  ADD NEXT LEVEL NODES TO TEMPORARY LIST
G       NODLST LNKLST SINK
        GO TO 10
C
C  BREAK THE LINKS AT THE CURRENT LEVEL,
C  ONE LINK TYPE AT A TIME.
G20     NODE-LINK
C
C  UPDATE THE LINK TYPE
        IF(LINK.EQ.ATTRIB) GO TO 30
        IF(LINK.EQ.DOWN) LINK=ATTRIB
C
C    THE 'RING' LINK MAY BE INCLUDED IN THE SEARCH BY
C    MODIFYING THE PREVIOUS TWO LINES AS FOLLOWS:
C       IF(LINK.EQ.RING) GO TO 30
C       IF(LINK.EQ.ATTRIB) LINK=RING
C       IF(LINK.EQ.DOWN) LINK=ATTRIB
C
        GO TO 6
C
C  REMOVE TOP NODE FROM TEMPORARY LIST
G30     NODLST+LNKLST-.1/RETURN 'NODE/5
G       COMPLETE
/       COMPLETE
```

PROGRAM 2, AS CODED IN FORTRAN FOR GIRS

```
      IMPLICIT INTEGER(A-Z)
      COMMON /LVARGS/ LVFUNC,LVVARG,LVVPOS,LVVTYP,LVVAL,
      1 LVVNVL,LVSKIP,LVVTR,LVVINC,LVNDXN,LVVALS(10),LVTYPE(10)
      COMMON /LVVSEQ/ LVSIZE,LVSEQ1,LVSEQ2,SEQSPC(   1)
      COMMON /LVTABL/ LVTSZE,LVEXTR,LVMAP(   1)
      COMMON /LVVTR5/ LVFILE,LVCMPR,NODESP(   1)
      1 /LVVTR6/ LISTSP(   1) /LVVTR7/ LINKSP(   1)/LVVTR8/ FLAGSP(   1)
      COMMON/LVRAND/LVKPRM,LVKS,LVKY,LVKDY,LVKDX,LVTEMP
      COMMON/LVVTR1/LVVSZE,LVVGSP,NODSPC(  50) /LVVTR2/
      1 LSTSPC(  50)/LVVTR3/LNKSPC(  50) /LVVTR4/FLGSPC(  50)
      COMMON /LINKS/ DOWN,ATTRIB,RING
      DATA LVTYPE /0,0,0,0,0,0,0,0,0,0/
      LVVSZE=  50
      LVFILE=23
      LVCMPR=0
      LVSIZE=    1
      LVSKIP=0
      LVKPRM= 3
      LVTSZE=   1
      LVVNVL=1
      LVVPOS=1
      LVVTYP=3
      LVNDXN=0
      LVEXTR=   0
      CALL ASSIGN(23,'RK0:GRAPH.BIN',13,'OLD')
      CALL LVFECH(LVFILE)
      GO TO 25001
25000 CONTINUE
      CALL ASSIGN(25,'RK0:GRAPH1.BCD',14,'NEW')
      CALL ASSIGN(27,'RK0:GRAPH1.BIN',14,'NEW')
      READ(23) HEAD,N1,N2,N3,N4,N5,N6,N7,N8,N9,N10,DOWN,ATTRIB,
      1 RING
C
C  THIS ROUTINE READS IN A GRAPH AND IS A DRIVER TO DELETE THE
C  SUBSTRUCTURE BELOW N6 AND THEN SAVE NEW GRAPH
C
      CALL DELNOD(N6)
C
C  OUTPUT MODIFIED GRAPH TO DISK
      CALL LVDUMP(0,0,27)
      WRITE(27) HEAD,N1,N2,N3,N4,N5,N6,N7,N8,N9,N10,DOWN,ATTRIB,
      1 RING
C
C  OUTPUT GRAPH IN BCD FORMAT TO BE PRINTED
      CALL LVDUMP(1,50,25)
      STOP
25001 CONTINUE
      GO TO 25000
      END
```

```
      SUBROUTINE DELNOD(NOD)
      IMPLICIT INTEGER(A-Z)
      COMMON /LVARGS/ LVFUNC,LVVARG,LVVPOS,LVVTYP,LVVAL,
      1 LVVNVL,LVSKIP,LVVTR,LVVINC,LVNDXN,LVVALS(10),LVTYPE(10)
      COMMON /LINKS/ DOWN,ATTRIB,RING
      CALL LVGRN(NODLST)
      CALL LVGRN(LNKLST)
      GO TO 25001
25000 CONTINUE
C
C  DELETE ENTIRE STRUCTURE BELOW 'NOD', ONE LEVEL AT A TIME,
C  ('BREADTH FIRST' SEARCH) THUS ELIMINATING THE POSSIBILITY
C  OF INFINITE LOOPS.
C
      NODE=NOD
 5    LINK=DOWN
C
C  PICK UP ALL POINTERS AT ONE LEVEL
 6    I=0
10    CONTINUE
C10   NODE+LINK,''I=I+1'/20'SINK
      LVVAL=NODE
      LVVARG=LVVAL
      LVFUNC=LINK
      CALL LVFIND
      LVVTYP=0
      I=I+1
      LVVPOS=  I
      CALL LVFNV(LV  1,LV  2,LV  3,LV  4)
      IF(LVVTR .EQ. -1) GO TO    20
      SINK
      1  =LVVAL
C
C  ADD NEXT LEVEL NODES TO TEMPORARY LIST
C     NODLST LNKLST SINK
      LVVAL=NODLST
      LVVARG=LVVAL
      LVFUNC=LNKLST
      CALL LVFIND
      LVVALS(1)=SINK
      CALL LVNSRT
      GO TO 10
C
C  BREAK THE LINKS AT THE CURRENT LEVEL,
C  ONE LINK TYPE AT A TIME.
20    CONTINUE
C20   NODE-LINK
      LVVAL=NODE
      LVVARG=LVVAL
      LVFUNC=LINK
      CALL LVDLET
C
C  UPDATE THE LINK TYPE
      IF(LINK.EQ.ATTRIB) GO TO 30
      IF(LINK.EQ.DOWN) LINK=ATTRIB
C
C    THE 'RING' LINK MAY BE INCLUDED IN THE SEARCH BY
C    MODIFYING THE PREVIOUS TWO LINES AS FOLLOWS:
C       IF(LINK.EQ.RING) GO TO 30
C       IF(LINK.EQ.ATTRIB) LINK=RING
C       IF(LINK.EQ.DOWN) LINK=ATTRIB
C
      GO TO 6
```

```
C
C  REMOVE TOP NODE FROM TEMPORARY LIST
30      CONTINUE
C30     NODLST+LNKLST-.1/RETURN 'NODE/5
        LVVAL=NODLST
        LVVARG=LVVAL
        LVFUNC=LNKLST
        CALL LVFIND
        CALL LVFNV(LV  5,LV  6,LV  7,LV  8)
        LVNDXN=1
        CALL LVDLET
        IF(LVVTR .EQ. -1) RETURN
        NODE
        1  =LVVAL
        IF(LVVTR .NE. -1) GO TO     5
        RETURN
25001   CONTINUE
        LV  1=0
        LV  2=0
        LV  3=0
        LV  4=0
        LV  5=0
        LV  6=0
        LV  7=0
        LV  8=0
        GO TO 25000
        END
```

CONTENTS OF GIRS BUFFER AFTER PROGRAM 2 COMPLETION

GIRS MEMORY DUMP (IN OCTAL)

REGASP= 8
MEMSZE= 50 PRIME= 3 SEED= 1 NROW= 3 KDNODE= 4 TEMP= 3
SEGSIZE= 1 MAPSIZE= 1

LOC	NODSPC	LSTSPC	LNKSPC	FLGSPC	OCTAL COUNTER
1	5	40	40	140	1
2	62	6	0	0	2
3	21	14	0	0	3
4	36	41	0	0	4
5	30	3	5	115	5
6	2	13	0	0	6
7	61	62	0	0	7
8	54	53	0	0	10
9	53	16	0	0	11
10	12	20	20	140	12
11	6	21	0	0	13
12	3	23	0	0	14
13	40	50	15	114	15
14	11	24	0	0	16
15	33	25	0	0	17
16	20	26	12	140	20
17	13	3	0	0	21
18	21	1	22	354	22
19	14	32	0	0	23
20	16	33	0	0	24
21	17	34	0	0	25
22	21	12	51	354	26
23	37	37	37	140	27
24	30	2	50	111	30
25	32	36	0	0	31
26	23	31	0	0	32
27	24	17	0	0	33
28	25	44	0	0	34
29	55	45	0	0	35
30	31	4	0	0	36
31	50	50	27	140	37
32	27	22	1	140	40
33	4	43	0	0	41
34	30	1	42	115	42
35	41	54	0	0	43
36	34	55	0	0	44
37	35	56	0	0	45
38	56	57	0	0	46
39	57	61	0	0	47
40	21	27	30	354	50
41	40	12	26	110	51
42	40	20	52	114	52
43	10	11	0	0	53
44	43	10	0	0	54
45	44	35	0	0	55
46	45	46	0	0	56
47	46	47	0	0	57
48	40	37	60	114	60
49	47	7	0	0	61
50	7	2	0	0	62

PROGRAM 3, AS CODED IN GIRL

```
 300 PRINT COMMENTS NOSAVE
G       DEFINE QUEUE,NAME,SENIOR,PROCES,CURENT,TABLE,SEQVEN,IO,FILREQ,
G     + INPUT,PERIPH,MEMORY,READY,BLOKED,DISK,TAPE,OTHER,AVAIL,
G     1 WAITDK,WAITTP,TEMP,END
        COMMON QUEUE,NAME,SENIOR,PROCES,CURENT,TABLE,SEQVEN,IO,CPURUN,
        1 FILREQ,TIME,AVAIL,PERIPH,MEMORY,READY,BLOKED,WAITDK,WAITTP
        COMMON/IOFILE/ DISK,TAPE,OTHER,END
        DIMENSION IFILES(12)
C
C     ASSIGN UNIQUE RANDOM NUMBERS TO LANGUAGE OPERATORS
C
C
        DATA ITAPE,IPRNTR,ICDRDR,DSK1FA,DSK1FB,DSK1FC,DSK2FD,
        1 DSK2FE,DSK2FF,BLANK,IEND,IDISK,IOTHER
        1 /2HTP,2HPR,2HCD,2HDA,2HDB,2HDC,2HDD,2HDE,2HDF,1H ,2HEN,
        1 2HDK,2HOT/
C
G       EXECUTE
C
C     THE LANGUAGE USED IS THE GRAPH INFORMATION RETRIEVAL LANGUAGE
C     PROGRAM WRITTEN BY IRVING S. ZARITSKY
C
C     OPERATION PRIMITIVES
C
C     INSERT FUNCTION (SOURCE NODE OR ARGUMENT, LINK, SINK NODE OR VALUE)
C        A B C
C
C     INSERT MULTI-VALUED LISTS (FOR NONDETERMINISMS OR DYNAMICALLY
C     CHANGING ARRAY LENGTHS)
C        A B (C,D,E,F)
C
C     REPLACE THE I'TH VALUE IN THE MULTI-VALUED LIST WITH VALUE G
C        A B-.I G
C
C     MAKE THE I'TH VALUE IN THE MULTI-VALUED LIST H
C        A B.I H
C
C     RETRIEVE THE FIRST VALUE ASSOCIATED WITH SOURCE NODE A AND LINK B
C        A+B
C
C     RETRIEVE THE I'TH VALUE ASSOCIATED WITH SOURCE NODE A AND LINK B
C        A+B.I
C
C     DELETE ENTIRE FUNCTION (MULTI-VALUED LIST)
C        A-B
C
C     DELETE I'TH VALUE FROM MULTI-VALUED LIST
C        A+B-.I
C
C     NAMING OPERATION
C        X'Y
C
C     FAILURE-SUCCESS OF PREVIOUS OPERATION TRANSFER
C        ...../FAILURE/SUCCESS  OR  ...../F/ FALL THRU  OR  ..../SUCCESS
C
C     DATA   RANDOM NUMBER  INTEGER   HOLLERITH
C                 $              'NUMBER'  '//HOL DATA' OR '/8/HOL DATA' OR )VARIABLE
C
        TYPE 5, QUEUE,NAME,SENIOR,PROCES,CURENT,TABLE,SEQVEN,IO,FILREQ,
        1 INPUT,PERIPH,MEMORY,READY,BLOKED,DISK,TAPE,OTHER,AVAIL,
        1 WAITDK,WAITTP,TEMP
   5  FORMAT(1X,9I6,//)
C
```

```
C
C*** INITIALLIZATION
C
        TIME=0
        SNRITY=9999-TIME
C
C     ALL PERIPHERALS ARE INITIALLY AVAILABLE, INSERT INTO GRAPH
G        PERIPH AVAIL (_ITAPE,_ITAPE,_IPRNTR,_IPRNTR,_ICDRDR,_ICDRDR,
G     + _DSK1FA,_DSK1FB,_DSK1FC,_DSK2FD,_DSK2FE,_DSK2FF)
C
C     ALL OF MEMORY IS AVAILABLE, INSERT INTO GRAPH
G        MEMORY AVAIL ('50','50','50','100')
C
C     THE CPU IS INITIALLY FREE, CPURUN WILL POINT TO THE PROCESS WHICH
C     IS EXECUTING
        CPURUN=0
C     INPUT FORMATS
C
C     JOB NAME, NO. OF PROCESSES
    1   FORMAT(A2,1X,I2)
C
C     PRIORITY OF PROCESS, MEMORY REQUIREMENT
    2   FORMAT(I2,1X,I3)
C
C     FILE REQUIREMENTS
    3   FORMAT(11(A2,1X))
C
C     SEQUENCE; CPU TIME-I/O FILE-I/O TIME
    4   FORMAT(I3,1X,A2,1X,I3)
C
C     OUTPUT FORMATS
    6   FORMAT(///,A2,1X,I2)
    7   FORMAT(1H ,I2,1X,I3)
    8   FORMAT(1H ,11(A2,1X))
    9   FORMAT(1H ,I3,1X,A2,1X,I3)
C
C*** READ AND INSERT NEXT JOB AND ITS REQUIREMENTS INTO THE INPUT
C     QUEUE GRAPH
C
        CALL ASSIGN(5,'RK0:TERM.DAT',12)
        CALL ASSIGN(16,'RK1:TERM.OUT',12)
   40   READ(5,1) JOB,NPROC
        TYPE 6,JOB,NPROC
C
C     IS THIS THE LAST JOB?
        IF(JOB.EQ.BLANK) GO TO 100
C
C     INSERT JOBNAME INTO GRAPH
G        INPUT QUEUE $'NXTJOB
C
C     INSERT JOB NAME AND SENIORITY WRT THIS QUEUE
G        NXTJOB (NAME _JOB ,SENIOR 'SNRITY')
C
C     READ IN NEXT PROCESS REQUIREMENTS
        DO 45 I=1,NPROC
C
C     READ IN INITIAL PRIORITY AND MEMORY REQUIREMENTS
        READ(5,2) KPRIOR,MEMREQ
        TYPE 7,  KPRIOR,MEMREQ
C
C     CREATE PROCESS TABLE
G        NXTJOB PROCES $'NXPROC(CURENT 'KPRIOR', TABLE('KPRIOR','MEMREQ',
G     + 'I'))
C
C     DETERMINE WHICH PERIPHERAL DEVICES ARE NEEDED.
C     THERE ARE 12 PERIPHERAL DEVICES; TWO CARDS MAY HAVE TO BE READ IN.
```

```
C
        DO 44 L=1,2
        READ(5,3) (IFILES(J),J=1,11)
        TYPE 8,  (IFILES(J),J=1,11)
        DO 46 K=1,11
C
C     NO MORE FILES REQUESTED?
        IF(IFILES(K).EQ. BLANK) GO TO 48
C
C     INSERT FILE REQUIREMENTS INTO INPUT QUEUE GRAPH
G       NXPROC FILREQ _*IFILES(K)
        IFILES(K)=BLANK
  46  CONTINUE
  44  CONTINUE
C
C     READ IN SEQUENCE OF CPU AND I/O EVENTS
  48  READ(5,4) CPUTIM,IOFILE,IOTIME
        TYPE 9,  CPUTIM,IOFILE,IOTIME
C
C
        IF(IOFILE.EQ.IOTHER) IOFIL=OTHER
        IF(IOFILE.EQ. ITAPE) IOFIL=TAPE
        IF(IOFILE.EQ. IDISK) IOFIL=DISK
        IF(IOFILE.EQ.  IEND) IOFIL=END
C
C     INSERT SEQUENCE OF EVENTS INTO INPUT GRAPH
G       NXPROC SEQVEN ('CPUTIM','IOTIME',IOFIL)
C
C     IS THIS SEQUENCE OVER?
        IF(IOFILE.NE.  IEND) GO TO 48
  45  CONTINUE
C     RETURN TO THE BEGINNING TO READ IN THE NEXT JOB.
        GO TO 40
C
C*** SORT PERIPHERAL QUEUE ACCORDING TO INITIAL PRIORITY AND SENIORITY.
C     PERIPHERAL QUEUE IS INITIALLY EMPTY
C
C     LOOK AT PRIORITY AND SENIORITY OF OLDEST PROCESS OF EACH JOB FROM
C     INPUT QUEUE
 100  I=0
G  61  INPUT +QUEUE.'I=I+1'/70'SINK+(PROCES+CURENT'KURPRI,SENIOR'NSNIOR)
        PRIOR2=(100*KURPRI)+NSNIOR
        J=0
G  67  PERIPH+QUEUE.'J=J+1'/66+(PROCES+CURENT'KURPRI,SENIOR'NSNIOR)
        PRIOR1=(100*KURPRI)+NSNIOR
        IF(PRIOR2.LE.PRIOR1) GO TO 67
C
C     INSERT PROCESS INTO J'TH POSITION IN PERIPHERAL QUEUE AND REPLACE
C     PREVIOUS QUEUE SENIORITY WITH PERIPHERAL QUEUE SENIORITY.
G  66  PERIPH QUEUE.J SINK SENIOR-.1 'SNRITY'/61/61
C
C     TYPE OUT RESOURCE STATUS
  70  CALL DUMP
C
C*** ALLOCATE PERIPHERAL RESOURCES ACCORDING TO PRIORITY AND SENIORITY
C
        I=0
C     TEMP LIST HOLDS PERIPHERALS FROM ALLOCATION QUEUE WHICH WILL BE PUT
C     BACK IF REQUEST CANNOT BE SATISFIED
C
C     CLEAR TEMP LIST
G  71  TEMP-AVAIL
C
C     LOOK AT PERIPHERAL REQUIREMENTS OF NEXT PROCESS IN PERIPHERAL
C     ALLOCATION QUEUE
G       PERIPH+QUEUE.'I=I+1'/72'SINK+PROCES'FILIST+CURENT'KURPRI
```

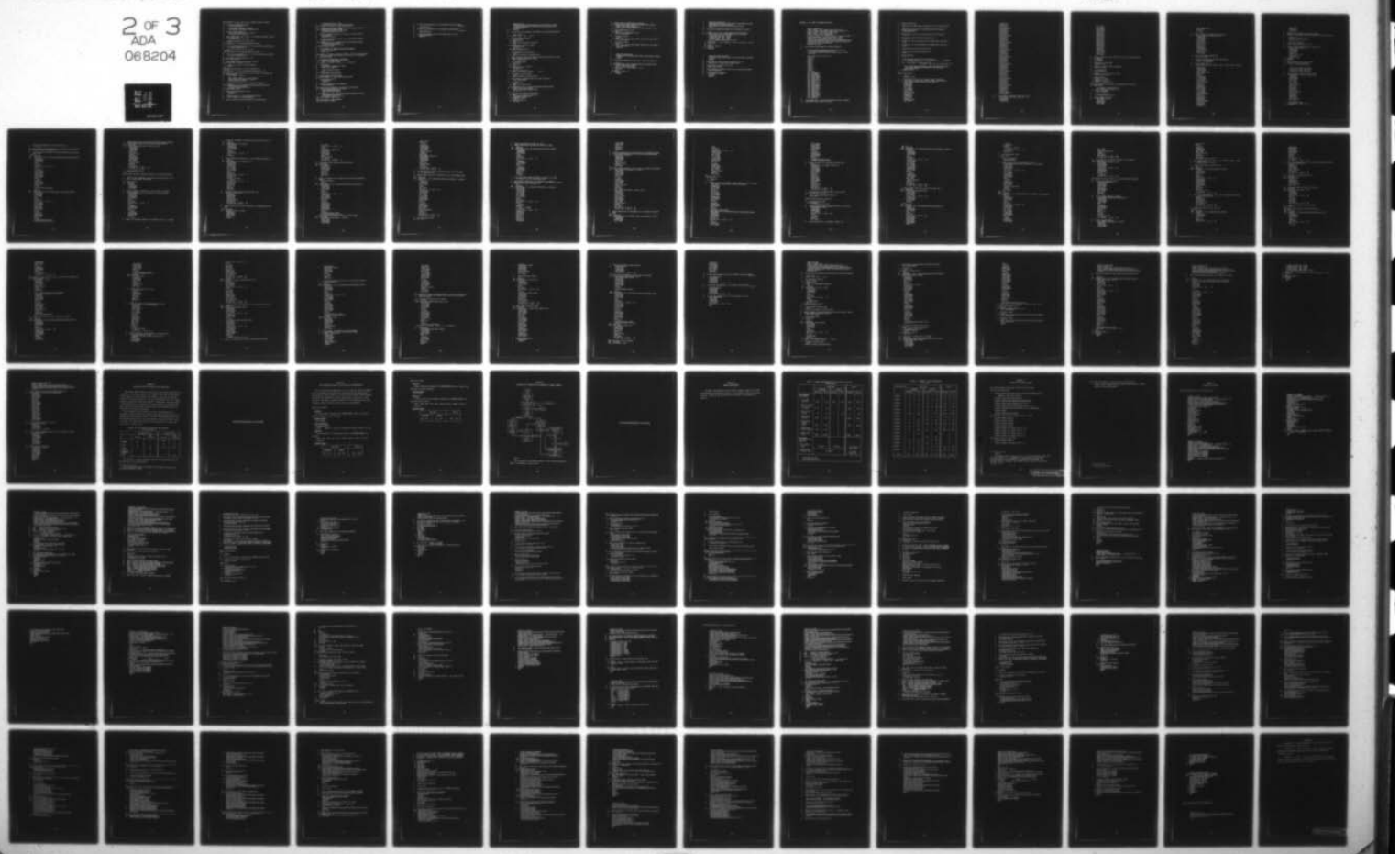
```
C
C     SEARCH THE PERIPHERAL REQUEST LIST OF PROCESS 'FILIST'.  IF THE END
C     OF THE LIST IS REACHED, THE REQUEST HAS BEEN SATISFIED, GO TO 78
          J=0
G   73 FILIST+FILREQ.'J=J+1'/78'NXFIL
C
C     COMPARE REQUESTED PERIPHERAL NXFIL WITH PERIPHERAL AVAILABLE LIST
          K=0
G   76 PERIPH+AVAIL(.'K=K+1'/75=NXFIL/76,-.K)
C
C     PERIPHERAL HAS BEEN MATCHED, PLACE ON TEMP LIST
G      TEMP AVAIL _NXFIL/73/73
C
C     REQUEST CANNOT BE MET, RETURN PERIPHERALS TO PERIPHERALS AVAILABLE
C     LIST
    75 M=0
G   77 TEMP+AVAIL.'M=M+1'/71'KTMPFL
G         PERIPH AVAIL _KTMPFL/77/77
C
C     REMOVE PROCESS FROM PERIPHERAL ALLOCATION QUEUE
G   78 PERIPH+QUEUE-.I
          I=I-1
C
C     INSERT THIS PROCESS INTO THE MEMORY ALLOCATION QUEUE ACCORDING TO
C     PRIORITY
          PRIOR1=KURPRI*100
          K=0
G   74 MEMORY+QUEUE.'K=K+1'/79+(SENIOR'NSNIOR,PROCES+CURENT'KURPRI)
          PRIOR2=(KURPRI*100)+NSNIOR
          IF(PRIOR2.GE.PRIOR1) GO TO 74
C     RESET PRIORITIES OF PROCESS IN MEMORY ALLOCATION QUEUE
G 79   FILIST(+TABLE'INITPR,CURENT-.1 'INITPR')
G       MEMORY QUEUE.K SINK/71/71
C
C     ALL PROCESSES WHICH CAN BE SATISFIED HAVE BEEN GIVEN PERIPHERALS
C     TYPE OUT RESOURCE STATUS
C
C     UPDATE CURRENT PRIORITY OF PROCESSES STILL IN THE PERIPHERAL QUEUE
   72   I=0
G   81 PERIPH+QUEUE.'I=I+1'/83+PROCES(+CURENT'KURPRI,CURENT-.1 'KURPRI+1'
G     + /81/81)
C
C     TYPE OUT RESOURCE STATUS
   83   CALL DUMP
C
C     MEMORY IS ALLOCATED ON A FIRST FIT BASIS
C     SEARCH MEMORY QUEUE, PROCESSES ARE IN PRIORITY ORDER.
          I=0
G   80 MEMORY+QUEUE.'I=I+1'/82'SINK+PROCES'SINK2+TABLE.2'MEMREQ
C
C     REJECT ANY PROCESS REQUESTING MEMORY IN EXCESS OF 100 WORDS.
          IF(MEMREQ.GT.100) CALL PURGE(MEMORY,SINK)
C
C     SEARCH MEMORY AVAILABLE LIST TO DETERMINE IF THE MEMORY
C     REQUIREMENTS OF THE PROCESS CAN BE MET.  IF SO, DELETE THAT MEMORY
C     BLOCK FROM THE MEMORY AVAILABLE LIST
          J=0
G   86 MEMORY+(AVAIL(.'J=J+1'/80'MEMLFT<MEMREQ//86,-.J),QUEUE-.I)
          I=I-1
C
C     A BLOCK OF MEMORY WHICH IS LARGE ENOUGH FOR THE PROCESS HAS BEEN
C     FOUND, MODIFY PROCESS TABLE SO THAT MEMORY REQUIREMENT BECOMES
G         SINK2 TABLE-.2 MEMLFT
C
C*** INSERT PROCESS INTO READY LIST AND RESET ITS PRIORITY AND SENIORITY
G         SINK2(+TABLE'INITPR,CURENT-.1 'INITPR')
```

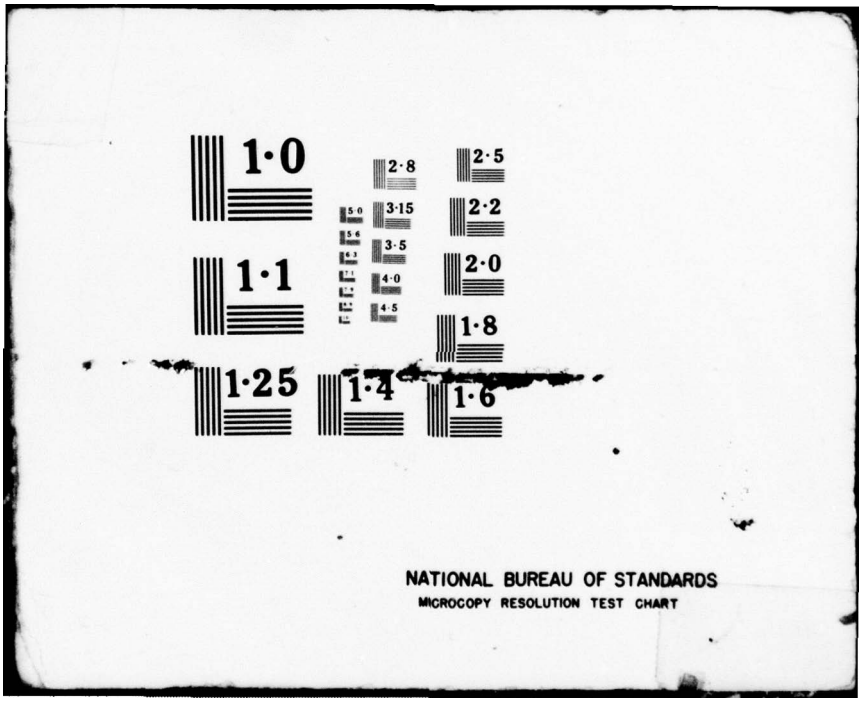
```
G       READY QUEUE SINK SENIOR-.1 'SNRITY'/80/80
C
C     UPDATE PRIORITIES OF ALL PROCESSES LEFT IN THE MEMORY ALLOCATION
C     QUEUE
  82   M=0
G  84 MEMORY+QUEUE.'M=M+1'/90+PROCES(+TABLE'INITPR,CURENT-.1'INITPR')
C
C*** SORT READY LIST
  90   I=0
         KSWICH=0
  91   I=I+1
C
C     COMPARE I'TH AND I+1'ST PRIORITY VALUES, SWITCH IF I+1'ST IS LARGER
G        READY+QUEUE.I+(PROCES+CURENT'KURPRI,SENIOR'NSNIOR)
         PRIOR1=(100*KURPRI)+NSNIOR
G        READY+QUEUE.'I+1'/92'SINK+(PROCES+CURENT'KURPRI,SENIOR'NSNIOR)
         PRIOR2=(100*KURPRI)+NSNIOR
         IF(PRIOR1.GE.PRIOR2) GO TO 91
C
C     SWITCH I'TH AND I+1'ST POSITIONS, SET SWITCH FLAG
         KSWICH=1
G        READY(+QUEUE-.'I+1',QUEUE.I SINK/91/91)
C
C     KEEP SORTING IF A CHANGE WAS MADE ON THE LAST PASS
  92   IF(KSWICH.EQ.1) GO TO 90
C
C     READY LIST HAS BEEN SORTED, REPORT OUT.
         CALL DUMP
C
C*** BEGIN CPU ALLOCATION ALGORITHM
C
C     IS A PROCESS EXECUTING?
 200 LEASTM=1000
         IF(CPURUN.NE.0) GO TO 204
C
C     TRANSFER HIGHEST PRIORITY PROCESS FROM READY LIST INTO CPU
G        READY+QUEUE/205'CPURUN -.1
C
C     UPDATE PRIORITIES OF ALL PROCESSES IN READY LIST
 204   I=0
G 381 READY+QUEUE.'I=I+1'/205+PROCES(+CURENT'KURPRI,CURENT-.1'KURPRI+1'
G     + /381/381)
C
C     SEARCH BLOCKED-FOR-I/O LIST AND CPU FOR LEAST TIME
G 205 CPURUN+PROCES/206+SEQVEN'LEASTM
  206 I=0
G 210 IO+BLOKED.'I=I+1'/220+PROCES+SEQVEN'IOLSTM<LEASTM/210
         LEASTM=IOLSTM
         GO TO 210
C
C     TYPE OUT RESOURCE STATUS
  220 CALL DUMP
C
C     UPDATE TIME, SENIORITY
         TIME=TIME+LEASTM
         SNRITY=9999-TIME
C
C     REDUCE TIME OF CPU AND I/O PROCESSES BY LEASTM
G        CPURUN+PROCES/224(+SEQVEN'ITIME,SEQVEN-.1'ITIME-LEASTM')
 224   I=0
G 225 IO+BLOKED.'I=I+1'/230+PROCES(+SEQVEN'ITIME,SEQVEN-.1'ITIME-LEASTM'
G     + /225/225)
C
C     HAS THE PROCESS IN EXECUTION FINISHED ITS TIME SEQUENCE
G 230 CPURUN +PROCES/300+SEQVEN='0'/300
C
```

AD-A068 204 DAVID W TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CE--ETC F/G 9/2
GIRS (GRAPH INFORMATION RETRIEVAL SYSTEM) USERS MANUAL.(U)
APR 79 I S ZARITSKY

UNCLASSIFIED DTNSRDC-79/036 NL

2 OF 3
ADA
068204




1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·5
2·8
3·15
3·5
4·0
4·5
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

```
C*** PROCESS IN CPU WILL SWITCH TO I/O UNLESS FINISHED OR BLOCKED
C
C     POP STACK OF SEQUENCE OF EVENTS
G        CPURUN+PROCES+SEQVEN-.1
C
C     IS THIS PROCESS COMPLETELY FINISHED?
G        CPURUN+PROCES'SINK+SEQVEN.2'MEDIA=END//400
C
C     RESET TO CURRENT PRIORITY
G        SINK(+TABLE'INITPR,CURENT-.1 'INITPR')
         IF(MEDIA.EQ.OTHER) GO TO 240
C
C     NEXT SEQUENCE USES TAPE OR DISK, IF THE APPROPIATE CHANNEL IS BUSY,
C     PUT IN WAIT STATE
         IF(MEDIA.EQ.TAPE ) GO TO 250
C
C     SEARCH BLOCKED-DOING-I/O LIST FOR DISK
            I=0
G 235 IO+BLOKED.'I=I+1'/240+PROCES+SEQVEN.2=DISK/235
C
C     DISK IS ON BLOCKED-DOING-I/O LIST, PLACE ON WAITING-FOR-DISK QUEUE
G        IO WAITDK CPURUN/260/260
C
C     SEARCH BLOCKED-DOING-I/O LIST FOR TAPE
  250 I=0
G 255 IO+BLOKED.'I=I+1'/240+PROCES+SEQVEN.2=TAPE/255
C
C     TAPE IS ON BLOCKED-DOING-I/O LIST, PLACE ON WAITING-FOR-TAPE QUEUE
G        IO WAITTP CPURUN/260/260
C
C     PLACE MEDIA ON BLOCKED-DOING-I/O LIST
G 240 IO BLOKED CPURUN
C
C     PLACE HIGHEST PRIORITY PROCESS INTO EXECUTION
G 260 READY+QUEUE /290'CPURUN ~.1//300
C
C        READY LIST IS EMPTY, NO JOB IS RUNNING
  290 CPURUN=0
C
C*** SEARCH BLOCKED-DOING-I/O LIST FOR COMPLETED I/O
  300 I=0
G 310 IO+BLOKED.'I=I+1'/350'SINK+PROCES'SINK2+SEQVEN='0'/310
C
C*** PROCESS IS TO BE TAKEN OFF THE I/O LIST AND PUT ON THE READY LIST
C     UPDATE SENIORITY
G        SINK SENIOR-.1 'SNRITY'
C
C        RESET CURRENT PRIORITY TO INITIAL PRIORITY
G        SINK2(+TABLE'INITPR,CURENT-.1 'INITPR')
         PRIOR1=(INITPR*100)+SNRITY
C
C     PLACE PROCESS INTO READY LIST IN POSITION WRT PRIORITY
            J=0
G 320 READY+QUEUE.'J=J+1'/340+(SENIOR'NSNIOR,PROCES+CURENT'KURPRI)
         PRIOR2=(KURPRI*100)+NSNIOR
         IF(PRIOR2.GE.PRIOR1) GO TO 320
G 340 READY QUEUE.J SINK
C
C     REMOVE FROM BLOCKED-DOING I/O LIST
G        IO+BLOKED-.I
         I=I-1
C
C     REMOVE MEDIA AND I/O TIME FROM SEQUENCE OF EVENTS
G        SINK2 +SEQVEN-.(1,1'MEDIA=OTHER//310)
C
C     CHECK I/O WAITING LISTS AND TRANSFER TO I/O-BLOCKED LIST
```

```
        IF(MEDIA.EQ.TAPE) GO TO 345
G       IO(+WAITDK/310'SINK3-.1,BLOKED SINK3/310/310)
G 345 IO(+WAITTP/310'SINK3-.1,BLOKED SINK3/310/310)
C
C       COMPARE PRIORITIES OF PROCESS WITH FIRST PROCESS IN READY LIST
G 350 CPURUN+PROCES/200+CURENT'KURPR1
G       READY+QUEUE/200+PROCES+CURENT'KURPR2
        IF(KURPR1.GE.KURPR2) GO TO 200
C
C    PRIORITY OF PROCESS IN READY LIST IS HIGHER THAN CPU, SWITCH.
        KTEMP=CPURUN
G       READY+QUEUE'CPURUN-.1
C
C    PUT DISPLACED PROCESS INTO READY LIST IN POSITION WRT PRIORITY
        PRIOR1=KURPR1*100
        I=0
G 370 READY+QUEUE."I=I+1"/375+(SENIOR'NSNIOR,PROCES+CURENT'KURPRI)
        PRIOR2=(KURPRI*100)+NSNIOR
        IF(PRIOR2.GE.PRIOR1) GO TO 370
G 375 READY QUEUE.I KTEMP
C
C    UPDATE PRIORITY AND SENIORITY OF OLD CPU PROCESS
G       KTEMP(SENIOR-.1 "SNRITY",+PROCES(+TABLE'INITPR,
G    + CURENT-.1 "INITPR-1"))
        GO TO 200
C
C*** PROCESS IS FINISHED, REALLOCATE RESOURCES, DELETE THAT PROCESS FROM
C    INPUT QUEUE, INSERT NEXT PROCESS OF THAT JOB INTO THE PERIPHERAL
C    ALLOCATION QUEUE.
C
C    TYPE NAME OF PROCESS WHICH IS FINISHED
G 400 CPURUN+(NAME'JOBNAM,PROCES+TABLE.3'NPROC)
        TYPE 10, TIME,JOBNAM,NPROC
  10  FORMAT(///,5X,'TIME  ',I3,A2,'.'I1,' IS FINISHED')
C
C    RETURN MEMORY
G       CPURUN+PROCES'NODE+TABLE.2'MEMREQ
G       MEMORY AVAIL "MEMREQ"
C
C       RETURN PERIPHERAL DEVICES
        I=0
G 410 NODE+FILREQ."I=I+1"/420'NFILE
G       PERIPH AVAIL _NFILE/410/410
C
C    REMOVE PROCESS FROM INPUT QUEUE
G 420 CPURUN+PROCES-.1-(CURENT,TABLE,SEQVEN,FILREQ)
C
C    REMOVE PROCESS FROM CPU
        KTEMP=CPURUN
        CPURUN=0
C
C    DOES THIS JOB HAVE ANY MORE PROCESSES?
G       KTEMP+PROCES/500
C
C*** PLACE THE NEXT PROCESS OF THIS JOB INTO THE PERIPHERAL
C    ALLOCATION QUEUE ACCORDING TO PRIORITY
G       KTEMP+PROCES+CURENT'KURPRI
        PRIOR1=(KURPRI*100)+SNRITY
        I=0
G 430 PERIPH+QUEUE."I=I+1"/440+(SENIOR'NSNIOR,PROCES+CURENT'KURPRI)
        PRIOR2=(KURPRI*100)+NSNIOR
        IF(PRIOR2.GE.PRIOR1) GO TO 430
G440  PERIPH QUEUE.I KTEMP/70/70
C
C*** THIS JOB HAS NO MORE PROCESSES
G 500 INPUT+QUEUE-.1 KTEMP
```

```
C
C     IF THERE ARE PROCESSES LEFT IN THE PERIPHERAL ALLOCATION QUEUE,
C                                                              GO TO 70
G        PERIPH+QUEUE//70
C
C     IF THERE ARE PROCESSES LEFT IN THE MEMORY ALLOCATION QUEUE,
C                                                              GO TO 83
G        MEMORY+QUEUE//83
C
C     IF THERE ARE PROCESSES LEFT IN THE READY QUEUE, GO TO 200
G        READY +QUEUE//200
G        COMPLETE
```

```
$       SUBROUTINE DUMP
        COMMON QUEUE,NAME,SENIOR,PROCES,CURENT,TABLE,SEQVEN,IO,CPURUN,
        1 FILREQ,TIME,AVAIL,PERIPH,MEMORY,READY,BLOKED,WAITDK,WAITTP
        DIMENSION IFILES(12)
        DATA BLANK/1H /
G       EXECUTE
C
C    THIS ROUTINE IS AN EXECUTIVE FOR TYPEING OUT ALL SYSTEM PARTICULARS
C
        TYPE 1,TIME
   1  FORMAT(///,5X,'TIME    ',I5,//)
C
C    TYPE PERIPHERALS AVAILABLE
        DO 20 I=1,12
  20   IFILES(I)=BLANK
        TYPE 2
   2   FORMAT(//,6X,'PERIPHERALS AVAILABLE')
        I=0
G  10 PERIPH+AVAIL.*I=I+1*/25 'IPERIP
        IFILES(I)=IPERIP
        GO TO 10
  25   TYPE 3,(IFILES(I),I=1,12)
   3   FORMAT(6X,12(A2,1X)///)
C
C    TYPE PERIPHERAL ALLOCATION QUEUE
        TYPE 4
   4   FORMAT(6X,'PERIPHERAL ALLOCATION QUEUE')
C
C    SEARCH PERIPHERAL ALLOCATION QUEUE FOR PROCESS PARTICULARS- JOB NAME
C    CURRENT PRIORITY, AND PROCESS NUMBER.
        CALL REPORT(PERIPH)
C
C    TYPE AVAILABLE MEMORY
        DO 90 I=1,4
  90   IFILES(I)=0
        I=0
G  91 MEMORY+AVAIL.*I=I+1*/95'MEM
        IFILES(I)=MEM
        GO TO 91
  95   TYPE 6,(IFILES(I),I=1,4)
   6   FORMAT(//,6X,'MEMORY AVAILABLE     ',4I5,//)
C
C    TYPE MEMORY ALLOCATION QUEUE
        TYPE 7
   7   FORMAT(6X,'MEMORY ALLOCATION QUEUE')
C
C    SEARCH MEMORY ALLOCATION QUEUE FOR PROCESS PARTICULARS
        CALL REPORT(MEMORY)
C
C    TYPE READY LIST
        TYPE 8
   8   FORMAT(//,6X,'READY LIST')
        M=0
G  16 READY+QUEUE.*M=M+1*/17+(NAME'JOBNAM,PROCES+(CURENT'KURPRI,
G     + SEQVEN'TIMLFT,TABLE.3'NPROC))
        TYPE 11,JOBNAM,NPROC,KURPRI,TIMLFT
        GO TO 16
C
C    TYPE STATUS OF PROCESS BEING EXECUTED
  17   TYPE 9
   9   FORMAT(//,6X,'PROCESS IN EXECUTION')
        IF(CPURUN.GT.0) GO TO 30
C    NO PROCESS IS RUNNING
        TYPE 15
  15   FORMAT(8X,'NONE',//)
        GO TO 40
```

```
C
C     CPURUN POINTS TO PROCESS WHICH IS RUNNING
G   30 CPURUN+(SENIOR'NSNRTY,NAME'JOBNAM,PROCES+(CURENT'KURPRI,
G     + SEQVEN'TIMLFT,TABLE.3'NPROC))
         TYPE 11,JOBNAM,NPROC,KURPRI,TIMLFT
   11  FORMAT(6X,A2,'.',I1,2X,'PRIORITY ',I2,4X,'CPU TIME ',I2,//)
C
C     TYPE BLOCKED-DOING-I/O LIST
   40  TYPE 12
   12  FORMAT(6X,'BLOCKED-DOING-I/O LIST')
         CALL IORPRT(BLOKED)
C
C     TYPE DISK I/O QUEUE
         TYPE 13
   13  FORMAT(//,6X,'LIST OF JOBS BLOCKED, WAITING FOR DISK CHANNEL')
         CALL IORPRT(WAITDK)
C
C     TYPE TAPE I/O QUEUE
         TYPE 14
   14  FORMAT(//,6X,'LIST OF JOBS BLOCKED, WAITING FOR TAPE CHANNEL')
         CALL IORPRT(WAITTP)
G        COMPLETE



$        SUBROUTINE REPORT(NODE)
         COMMON QUEUE,NAME,SENIOR,PROCES,CURENT,TABLE,SEQVEN,IO,CPURUN,
         1 FILREQ
G        EXECUTE
C
C     THIS ROUTINE SEARCHES THE QUEUE NAMED BY NODE AND REPORTS OUT
C
         I=0
G   10 NODE+QUEUE/18.*I=I+1*/15+(SENIOR'NSNRTY,NAME'JOBNAM,PROCES+
G     + (CURENT'KURPRI,TABLE.3'NPROC))
         TYPE 5, JOBNAM,NPROC,KURPRI
    5   FORMAT(6X,A2,'.',I1,2X,'PRIORITY ',I2,/)
         GO TO 10
   18  TYPE 17
   17  FORMAT(8X,'NONE',//)
   15  CONTINUE
G        COMPLETE
```

```
$       SUBROUTINE IORPRT(LINK)
        COMMON QUEUE,NAME,SENIOR,PROCES,CURENT,TABLE,SEQVEN,IO,CPURUN
        COMMON/IOFILE/ DISK,TAPE,OTHER,END
        DATA ITAPE,IEND,IDISK,IOTHER/2HTP,2HEN,2HDK,2HOT/
G       EXECUTE
C
C    THIS ROUTINE SEARCHES AND REPORTS ON THE REQUESTED I/O QUEUE
C
        I=0
G  10 IO+LINK/18.*I=I+1*/15+(SENIOR'NSNRTY,NAME'JOBNAM,PROCES+
G    + (CURENT'KURPRI,TABLE.3'NPROC,SEQVEN('TIMLFT,.2'MEDIA)))
        IF(MEDIA.EQ.OTHER) MED  =IOTHER
        IF(MEDIA.EQ.TAPE ) MED  =ITAPE
        IF(MEDIA.EQ.DISK ) MED  =IDISK
        IF(MEDIA.EQ.END  ) MED  =IEND
        TYPE 12, JOBNAM,NPROC,KURPRI,TIMLFT,MED
        GO TO 10
  12  FORMAT(6X,A2,'.',I1,2X,'PRIORITY ',I2,4X,'I/O TIME LEFT ',I2,4X,
       1 A2,///)
  18  TYPE 17
  17  FORMAT(8X,'NONE',//)
  15  CONTINUE
G       COMPLETE
```

```
$       SUBROUTINE PURGE(IFILE,NODE)
        COMMON QUEUE,NAME,SENIOR,PROCES,CURENT,TABLE,SEQVEN,IO,CPURUN,
       1 FILREQ
G       EXECUTE
C
C    DELETE FROM THE GRAPH THE OLDEST PROCESS OF THE JOB
G       NODE+PROCES-.1-(CURENT,TABLE,FILREQ,SEQVEN)
C
C    WAS THAT THE LAST PROCESS OF THE JOB?
G     NODE+PROCES//RETURN
C
C    DELETE REMAINING INFORMATION ABOUT THE JOB (GARBAGE COLLECTION)
G       NODE-(SENIOR,NAME)
C
C    DELETE FROM SPECIFIED QUEUE
G       IFILE+QUEUE-.: CPURUN
G       COMPLETE
/       COMPLETE
```

PROGRAM 3, AS CODED IN FORTRAN FOR GIRS

```
      IMPLICIT INTEGER(A-Z)
      COMMON /LVARGS/ LVFUNC,LVVARG,LVVPOS,LVVTYP,LVVAL,
     1 LVVNVL,LVSKIP,LVVTR,LVVINC,LVNDXN,LVVALS(10),LVTYPE(10)
      COMMON /LVVSEQ/ LVSIZE,LVSEQ1,LVSEQ2,SEQSPC(    1)
      COMMON /LVTABL/ LVTSZE,LVEXTR,LVMAP(    1)
      COMMON /LVVTR5/ LVFILE,LVCMPR,NODESP(    1)
     1 /LVVTR6/ LISTSP(    1) /LVVTR7/ LINKSP(    1)/LVVTR8/ FLAGSP(    1)
      COMMON/LVRAND/LVKPRM,LVKS,LVKY,LVKDY,LVKDX,LVTEMP
      COMMON/LVVTR1/LVVSZE,LVVGSP,NODSPC( 300) /LVVTR2/
     1 LSTSPC( 300)/LVVTR3/LNKSPC( 300) /LVVTR4/FLGSPC( 300)
      COMMON QUEUE,NAME,SENIOR,PROCES,CURENT,TABLE,SEQVEN,IO,CPURUN,
     1 FILREQ,TIME,AVAIL,PERIPH,MEMORY,READY,BLOKED,WAITDK,WAITTP
      COMMON/IOFILE/ DISK,TAPE,OTHER,END
      DIMENSION IFILES(12)
C
C     ASSIGN UNIQUE RANDOM NUMBERS TO LANGUAGE OPERATORS
C
C
      DATA ITAPE,IPRNTR,ICDRDR,DSK1FA,DSK1FB,DSK1FC,DSK2FD,
     1 DSK2FE,DSK2FF,BLANK,IEND,IDISK,IOTHER
     1 /2HTP,2HPR,2HCD,2HDA,2HDB,2HDC,2HDD,2HDE,2HDF,1H ,2HEN,
     1 2HDK,2HOT/
C
      DATA LVTYPE /0,0,0,0,0,0,0,0,0,0/
      LVVSZE= 300
      LVFILE= 0
      LVCMPR=0
      LVSIZE=    1
      LVSKIP=1
      LVKPRM= 7
      LVTSZE=   1
      LVVNVL=1
      LVVPOS=1
      LVVTYP=3
      LVNDXN=0
      LVEXTR=   0
      CALL LVSETP
      CALL LVGRN(QUEUE )
      CALL LVGRN(NAME  )
      CALL LVGRN(SENIOR)
      CALL LVGRN(PROCES)
      CALL LVGRN(CURENT)
      CALL LVGRN(TABLE )
      CALL LVGRN(SEQVEN)
      CALL LVGRN(IO    )
      CALL LVGRN(FILREQ)
      CALL LVGRN(INPUT )
      CALL LVGRN(PERIPH)
      CALL LVGRN(MEMORY)
      CALL LVGRN(READY )
      CALL LVGRN(BLOKED)
      CALL LVGRN(DISK  )
      CALL LVGRN(TAPE  )
      CALL LVGRN(OTHER )
      CALL LVGRN(AVAIL )
      CALL LVGRN(WAITDK)
      CALL LVGRN(WAITTP)
      CALL LVGRN(TEMP  )
      CALL LVGRN(END   )
C
C     THE LANGUAGE USED IS THE GRAPH INFORMATION RETRIEVAL LANGUAGE
C     PROGRAM WRITTEN BY IRVING S. ZARITSKY
C
```

```
C     OPERATION PRIMITIVES
C
C     INSERT FUNCTION (SOURCE NODE OR ARGUMENT, LINK, SINK NODE OR VALUE)
C        A B C
C
C     INSERT MULTI-VALUED LISTS (FOR NONDETERMINISMS OR DYNAMICALLY
C     CHANGING ARRAY LENGTHS)
C        A B (C,D,E,F)
C
C     REPLACE THE I'TH VALUE IN THE MULTI-VALUED LIST WITH VALUE G
C        A B-.I G
C
C     MAKE THE I'TH VALUE IN THE MULTI-VALUED LIST H
C        A B.I H
C
C     RETRIEVE THE FIRST VALUE ASSOCIATED WITH SOURCE NODE A AND LINK B
C        A+B
C
C     RETRIEVE THE I'TH VALUE ASSOCIATED WITH SOURCE NODE A AND LINK B
C        A+B.I
C
C     DELETE ENTIRE FUNCTION (MULTI-VALUED LIST)
C        A-B
C
C     DELETE I'TH VALUE FROM MULTI-VALUED LIST
C        A+B-.I
C
C     NAMING OPERATION
C        X'Y
C
C     FAILURE-SUCCESS OF PREVIOUS OPERATION TRANSFER
C        ...../FAILURE/SUCCESS  OR  ...../F/ FALL THRU  OR  ..../SUCCESS
C
C     DATA   RANDOM NUMBER  INTEGER   HOLLERITH
C                 $              'NUMBER'  '//HOL DATA' OR '/8/HOL DATA' OR J(
C
        TYPE 5, QUEUE,NAME,SENIOR,PROCES,CURENT,TABLE,SEQVEN,IO,FILREQ,
        1 INPUT,PERIPH,MEMORY,READY,BLOKED,DISK,TAPE,OTHER,AVAIL,
        1 WAITDK,WAITTP,TEMP
    5  FORMAT(1X,9I6,//)
C
C
C*** INITIALLIZATION
C
        TIME=0
        SNRITY=9999-TIME
C
C     ALL PERIPHERALS ARE INITIALLY AVAILABLE, INSERT INTO GRAPH
C        PERIPH AVAIL (_ITAPE,_ITAPE,_IPRNTR,_IPRNTR,_ICDRDR,_ICDRDR,
C     + _DSK1FA,_DSK1FB,_DSK1FC,_DSK2FD,_DSK2FE,_DSK2FF)
        LVVAL=PERIPH
        LVVARG=LVVAL
        LVFUNC=AVAIL
        CALL LVFIND
        LVV 1=LVVAL
        LVV 2=LVFUNC
        LVV 3=LVVARG
        LVTYPE(1)=3
        LVVALS(1)=ITAPE
        CALL LVNSRT
        LVFUNC=LVV 2
        LVVARG=LVV 3
        CALL LVFIND
        LVTYPE(1)=3
        LVVALS(1)=ITAPE
        CALL LVNSRT
```

```
      LVFUNC=LVV 2
      LVVARG=LVV 3
      CALL LVFIND
      LVTYPE(1)=3
      LVVALS(1)=IPRNTR
      CALL LVNSRT
      LVFUNC=LVV 2
      LVVARG=LVV 3
      CALL LVFIND
      LVTYPE(1)=3
      LVVALS(1)=IPRNTR
      CALL LVNSRT
      LVFUNC=LVV 2
      LVVARG=LVV 3
      CALL LVFIND
      LVTYPE(1)=3
      LVVALS(1)=ICDRDR
      CALL LVNSRT
      LVFUNC=LVV 2
      LVVARG=LVV 3
      CALL LVFIND
      LVTYPE(1)=3
      LVVALS(1)=ICDRDR
      CALL LVNSRT
      LVFUNC=LVV 2
      LVVARG=LVV 3
      CALL LVFIND
      LVTYPE(1)=3
      LVVALS(1)=DSK1FA
      CALL LVNSRT
      LVFUNC=LVV 2
      LVVARG=LVV 3
      CALL LVFIND
      LVTYPE(1)=3
      LVVALS(1)=DSK1FB
      CALL LVNSRT
      LVFUNC=LVV 2
      LVVARG=LVV 3
      CALL LVFIND
      LVTYPE(1)=3
      LVVALS(1)=DSK1FC
      CALL LVNSRT
      LVFUNC=LVV 2
      LVVARG=LVV 3
      CALL LVFIND
      LVTYPE(1)=3
      LVVALS(1)=DSK2FD
      CALL LVNSRT
      LVFUNC=LVV 2
      LVVARG=LVV 3
      CALL LVFIND
      LVTYPE(1)=3
      LVVALS(1)=DSK2FE
      CALL LVNSRT
      LVFUNC=LVV 2
      LVVARG=LVV 3
      CALL LVFIND
      LVTYPE(1)=3
      LVVALS(1)=DSK2FF
      CALL LVNSRT
C
C     ALL OF MEMORY IS AVAILABLE, INSERT INTO GRAPH
C        MEMORY AVAIL ('50','50','50','100')
         LVVAL=MEMORY
         LVVARG=LVVAL
         LVFUNC=AVAIL
```

```
        CALL LVFIND
        LVV 1=LVVAL
        LVV 2=LVFUNC
        LVV 3=LVVARG
        LVVALS(1)=50
        LVTYPE(1)=1
        CALL LVNSRT
        LVFUNC=LVV 2
        LVVARG=LVV 3
        CALL LVFIND
        LVVALS(1)=50
        LVTYPE(1)=1
        CALL LVNSRT
        LVFUNC=LVV 2
        LVVARG=LVV 3
        CALL LVFIND
        LVVALS(1)=50
        LVTYPE(1)=1
        CALL LVNSRT
        LVFUNC=LVV 2
        LVVARG=LVV 3
        CALL LVFIND
        LVVALS(1)=100
        LVTYPE(1)=1
        CALL LVNSRT
C
C     THE CPU IS INITIALLY FREE, CPURUN WILL POINT TO THE PROCESS WHICH
C     IS EXECUTING
        CPURUN=0
C     INPUT FORMATS
C
C     JOB NAME, NO. OF PROCESSES
    1   FORMAT(A2,1X,I2)
C
C     PRIORITY OF PROCESS, MEMORY REQUIREMENT
    2   FORMAT(I2,1X,I3)
C
C     FILE REQUIREMENTS
    3   FORMAT(11(A2,1X))
C
C     SEQUENCE; CPU TIME-I/O FILE-I/O TIME
    4   FORMAT(I3,1X,A2,1X,I3)
C
C     OUTPUT FORMATS
    6   FORMAT(///,A2,1X,I2)
    7   FORMAT(1H ,I2,1X,I3)
    8   FORMAT(1H ,11(A2,1X))
    9   FORMAT(1H ,I3,1X,A2,1X,I3)
C
C*** READ AND INSERT NEXT JOB AND ITS REQUIREMENTS INTO THE INPUT
C     QUEUE GRAPH
C
        CALL ASSIGN(5,'RK0:TERM.DAT',12)
        CALL ASSIGN(16,'RK1:TERM.OUT',12)
   40   READ(5,1) JOB,NPROC
        TYPE 6,JOB,NPROC
C
C     IS THIS THE LAST JOB?
        IF(JOB.EQ.BLANK) GO TO 100
C
C     INSERT JOBNAME INTO GRAPH
C       INPUT QUEUE $'NXTJOB
        LVVAL=INPUT
        LVVARG=LVVAL
        LVFUNC=QUEUE
        CALL LVFIND
```

```
      CALL LVGRN(LVVALS(1))
      CALL LVNSRT
      NXTJOB
      1  =LVVAL
C
C     INSERT JOB NAME AND SENIORITY WRT THIS QUEUE
C        NXTJOB (NAME _JOB ,SENIOR 'SNRITY')
         LVVAL=NXTJOB
         LVVARG=LVVAL
         LVV 1=LVVARG
         LVFUNC=NAME
         CALL LVFIND
         LVTYPE(1)=3
         LVVALS(1)=JOB
         CALL LVNSRT
         LVVAL=LVV 1
         LVVARG=LVVAL
         LVFUNC=SENIOR
         CALL LVFIND
         LVVALS(1)=SNRITY
         LVTYPE(1)=1
         CALL LVNSRT
C
C     READ IN NEXT PROCESS REQUIREMENTS
         DO 45 I=1,NPROC
C
C     READ IN INITIAL PRIORITY AND MEMORY REQUIREMENTS
         READ(5,2) KPRIOR,MEMREQ
         TYPE 7,  KPRIOR,MEMREQ
C
C     CREATE PROCESS TABLE
C        NXTJOB PROCES $'NXPROC(CURENT 'KPRIOR', TABLE('KPRIOR','MEMREQ',
C     + 'I'))
         LVVAL=NXTJOB
         LVVARG=LVVAL
         LVFUNC=PROCES
         CALL LVFIND
         CALL LVGRN(LVVALS(1))
         CALL LVNSRT
         NXPROC
         1  =LVVAL
         LVVARG=LVVAL
         LVV 1=LVVARG
         LVFUNC=CURENT
         CALL LVFIND
         LVVALS(1)=KPRIOR
         LVTYPE(1)=1
         CALL LVNSRT
         LVVAL=LVV 1
         LVVARG=LVVAL
         LVFUNC=TABLE
         CALL LVFIND
         LVV 2=LVVAL
         LVV 3=LVFUNC
         LVV 4=LVVARG
         LVVALS(1)=KPRIOR
         LVTYPE(1)=1
         CALL LVNSRT
         LVFUNC=LVV 3
         LVVARG=LVV 4
         CALL LVFIND
         LVVALS(1)=MEMREQ
         LVTYPE(1)=1
         CALL LVNSRT
         LVFUNC=LVV 3
         LVVARG=LVV 4
```

```
      CALL LVFIND
      LVVALS(1)=I
      LVTYPE(1)=1
      CALL LVNSRT
C
C     DETERMINE WHICH PERIPHERAL DEVICES ARE NEEDED.
C     THERE ARE 12 PERIPHERAL DEVICES; TWO CARDS MAY HAVE TO BE READ IN.
C
      DO 44 L=1,2
      READ(5,3) (IFILES(J),J=1,11)
      TYPE 8,  (IFILES(J),J=1,11)
      DO 46 K=1,11
C
C     NO MORE FILES REQUESTED?
      IF(IFILES(K).EQ. BLANK) GO TO 48
C
C     INSERT FILE REQUIREMENTS INTO INPUT QUEUE GRAPH
C       NXPROC FILREQ _*IFILES(K)
      LVVAL=NXPROC
      LVVARG=LVVAL
      LVFUNC=FILREQ
      CALL LVFIND
      LVTYPE(1)=3
      LVVALS(1)=IFILES(K)
      CALL LVNSRT
      IFILES(K)=BLANK
 46   CONTINUE
 44   CONTINUE
C
C     READ IN SEQUENCE OF CPU AND I/O EVENTS
 48   READ(5,4) CPUTIM,IOFILE,IOTIME
      TYPE 9,  CPUTIM,IOFILE,IOTIME
C
C
      IF(IOFILE.EQ.IOTHER) IOFIL=OTHER
      IF(IOFILE.EQ. ITAPE) IOFIL=TAPE
      IF(IOFILE.EQ. IDISK) IOFIL=DISK
      IF(IOFILE.EQ.  IEND) IOFIL=END
C
C     INSERT SEQUENCE OF EVENTS INTO INPUT GRAPH
C       NXPROC SEQVEN ('CPUTIM','IOTIME',IOFIL)
      LVVAL=NXPROC
      LVVARG=LVVAL
      LVFUNC=SEQVEN
      CALL LVFIND
      LVV 1=LVVAL
      LVV 2=LVFUNC
      LVV 3=LVVARG
      LVVALS(1)=CPUTIM
      LVTYPE(1)=1
      CALL LVNSRT
      LVFUNC=LVV 2
      LVVARG=LVV 3
      CALL LVFIND
      LVVALS(1)=IOTIME
      LVTYPE(1)=1
      CALL LVNSRT
      LVFUNC=LVV 2
      LVVARG=LVV 3
      CALL LVFIND
      LVVALS(1)=IOFIL
      CALL LVNSRT
C
C     IS THIS SEQUENCE OVER?
      IF(IOFILE.NE.  IEND) GO TO 48
 45   CONTINUE
```

```
C     RETURN TO THE BEGINNING TO READ IN THE NEXT JOB.
         GO TO 40
C
C*** SORT PERIPHERAL QUEUE ACCORDING TO INITIAL PRIORITY AND SENIORITY.
C     PERIPHERAL QUEUE IS INITIALLY EMPTY
C
C     LOOK AT PRIORITY AND SENIORITY OF OLDEST PROCESS OF EACH JOB FROM
C     INPUT QUEUE
  100 I=0
 61      CONTINUE
C61    INPUT +QUEUE.'I=I+1'/70'SINK+(PROCES+CURENT'KURPRI,SENIOR'NSNIOR)
         LVVAL=INPUT
         LVVARG=LVVAL
         LVFUNC=QUEUE
         CALL LVFIND
         I=I+1
         LVVPOS=  I
         CALL LVFNV
         IF(LVVTR .EQ. -1) GO TO    70
         SINK
         1  =LVVAL
         LVVARG=LVVAL
         LVV 1=LVVARG
         LVFUNC=PROCES
         CALL LVFIND
         LVVARG=LVVAL
         LVFUNC=CURENT
         CALL LVFIND
         KURPRI
         1  =LVVAL
         LVVAL=LVV 1
         LVVARG=LVVAL
         LVFUNC=SENIOR
         CALL LVFIND
         NSNIOR
         1  =LVVAL
         PRIOR2=(100*KURPRI)+NSNIOR
         J=0
 67      CONTINUE
C67    PERIPH+QUEUE.'J=J+1'/66+(PROCES+CURENT'KURPRI,SENIOR'NSNIOR)
         LVVAL=PERIPH
         LVVARG=LVVAL
         LVFUNC=QUEUE
         CALL LVFIND
         J=J+1
         LVVPOS=  J
         CALL LVFNV
         IF(LVVTR .EQ. -1) GO TO    66
         LVVARG=LVVAL
         LVV 1=LVVARG
         LVFUNC=PROCES
         CALL LVFIND
         LVVARG=LVVAL
         LVFUNC=CURENT
         CALL LVFIND
         KURPRI
         1  =LVVAL
         LVVAL=LVV 1
         LVVARG=LVVAL
         LVFUNC=SENIOR
         CALL LVFIND
         NSNIOR
         1  =LVVAL
         PRIOR1=(100*KURPRI)+NSNIOR
         IF(PRIOR2.LE.PRIOR1) GO TO 67
C
```

```
C     INSERT PROCESS INTO J'TH POSITION IN PERIPHERAL QUEUE AND REPLACE
C     PREVIOUS QUEUE SENIORITY WITH PERIPHERAL QUEUE SENIORITY.
 66      CONTINUE
C66    PERIPH QUEUE.J SINK SENIOR-.1 'SNRITY'/61/61
          LVVAL=PERIPH
          LVVARG=LVVAL
          LVFUNC=QUEUE
          CALL LVFIND
          LVVPOS=  J
          CALL LVFNV
          LVVALS(1)=SINK
          LVNDXN=2
          CALL LVNSRT
          LVVARG=LVVAL
          LVFUNC=SENIOR
          CALL LVFIND
          CALL LVFNV
          LVVALS(1)=SNRITY
          LVTYPE(1)=1
          LVNDXN=1
          CALL LVNSRT
          IF(LVVTR .EQ. -1) GO TO    61
          IF(LVVTR .NE. -1) GO TO    61
C
C     TYPE OUT RESOURCE STATUS
  70   CALL DUMP
C
C*** ALLOCATE PERIPHERAL RESOURCES ACCORDING TO PRIORITY AND SENIORITY
C
          I=0
C     TEMP LIST HOLDS PERIPHERALS FROM ALLOCATION QUEUE WHICH WILL BE PUT
C     BACK IF REQUEST CANNOT BE SATISFIED
C
C     CLEAR TEMP LIST
 71      CONTINUE
C71    TEMP-AVAIL
          LVVAL=TEMP
          LVVARG=LVVAL
          LVFUNC=AVAIL
          CALL LVDLET
C
C     LOOK AT PERIPHERAL REQUIREMENTS OF NEXT PROCESS IN PERIPHERAL
C     ALLOCATION QUEUE
C        PERIPH+QUEUE.'I=I+1'/72'SINK+PROCES'FILIST+CURENT'KURPRI
          LVVAL=PERIPH
          LVVARG=LVVAL
          LVFUNC=QUEUE
          CALL LVFIND
          I=I+1
          LVVPOS=  I
          CALL LVFNV
          IF(LVVTR .EQ. -1) GO TO    72
          SINK
          1  =LVVAL
          LVVARG=LVVAL
          LVFUNC=PROCES
          CALL LVFIND
          FILIST
          1  =LVVAL
          LVVARG=LVVAL
          LVFUNC=CURENT
          CALL LVFIND
          KURPRI
          1  =LVVAL
C
C     SEARCH THE PERIPHERAL REQUEST LIST OF PROCESS 'FILIST'.  IF THE END
```

```
C     OF THE LIST IS REACHED, THE REQUEST HAS BEEN SATISFIED, GO TO 78
        J=0
 73     CONTINUE
C73    FILIST+FILREQ.*J=J+1*/78'NXFIL
        LVVAL=FILIST
        LVVARG=LVVAL
        LVFUNC=FILREQ
        CALL LVFIND
        J=J+1
        LVVPOS=  J
        CALL LVFNV
        IF(LVVTR .EQ. -1) GO TO    78
        NXFIL
        1  =LVVAL
C
C     COMPARE REQUESTED PERIPHERAL NXFIL WITH PERIPHERAL AVAILABLE LIST
        K=0
 76     CONTINUE
C76    PERIPH+AVAIL(.*K=K+1*/75=NXFIL/76,-.K)
        LVVAL=PERIPH
        LVVARG=LVVAL
        LVFUNC=AVAIL
        CALL LVFIND
        LVV 1=LVVAL
        LVV 2=LVFUNC
        LVV 3=LVVARG
        K=K+1
        LVVPOS=  K
        CALL LVFNV
        IF(LVVTR .EQ. -1) GO TO    75
        LVVARG=LVVAL
        LVVTR=-1
        IF(LVVAL .EQ. NXFIL
        1 ) LVVTR=1
        IF(LVVTR .EQ. -1) GO TO    76
        LVFUNC=LVV 2
        LVVARG=LVV 3
        CALL LVFIND
        LVVPOS=  K
        CALL LVFNV
        LVNDXN=1
        CALL LVDLET
C
C     PERIPHERAL HAS BEEN MATCHED, PLACE ON TEMP LIST
C      TEMP AVAIL _NXFIL/73/73
        LVVAL=TEMP
        LVVARG=LVVAL
        LVFUNC=AVAIL
        CALL LVFIND
        LVTYPE(1)=3
        LVVALS(1)=NXFIL
        CALL LVNSRT
        IF(LVVTR .EQ. -1) GO TO    73
        IF(LVVTR .NE. -1) GO TO    73
C
C     REQUEST CANNOT BE MET, RETURN PERIPHERALS TO PERIPHERALS AVAILABLE
C     LIST
   75 M=0
 77     CONTINUE
C77    TEMP+AVAIL.*M=M+1*/71'KTMPFL
        LVVAL=TEMP
        LVVARG=LVVAL
        LVFUNC=AVAIL
        CALL LVFIND
        M=M+1
        LVVPOS=  M
```

```
      CALL LVFNV
      IF(LVVTR .EQ. -1) GO TO    71
      KTMPFL
      1  =LVVAL
C     PERIPH AVAIL _KTMPFL/77/77
      LVVAL=PERIPH
      LVVARG=LVVAL
      LVFUNC=AVAIL
      CALL LVFIND
      LVTYPE(1)=3
      LVVALS(1)=KTMPFL
      CALL LVNSRT
      IF(LVVTR .EQ. -1) GO TO    77
      IF(LVVTR .NE. -1) GO TO    77
C
C     REMOVE PROCESS FROM PERIPHERAL ALLOCATION QUEUE
 78   CONTINUE
C78   PERIPH+QUEUE-.I
      LVVAL=PERIPH
      LVVARG=LVVAL
      LVFUNC=QUEUE
      CALL LVFIND
      LVVPOS=  I
      CALL LVFNV
      LVNDXN=1
      CALL LVDLET
      I=I-1
C
C     INSERT THIS PROCESS INTO THE MEMORY ALLOCATION QUEUE ACCORDING TO
C     PRIORITY
      PRIOR1=KURPRI*100
      K=0
 74   CONTINUE
C74   MEMORY+QUEUE.'K=K+1'/79+(SENIOR'NSNIOR,PROCES+CURENT'KURPRI)
      LVVAL=MEMORY
      LVVARG=LVVAL
      LVFUNC=QUEUE
      CALL LVFIND
      K=K+1
      LVVPOS=  K
      CALL LVFNV
      IF(LVVTR .EQ. -1) GO TO    79
      LVVARG=LVVAL
      LVV 1=LVVARG
      LVFUNC=SENIOR
      CALL LVFIND
      NSNIOR
      1  =LVVAL
      LVVAL=LVV 1
      LVVARG=LVVAL
      LVFUNC=PROCES
      CALL LVFIND
      LVVARG=LVVAL
      LVFUNC=CURENT
      CALL LVFIND
      KURPRI
      1  =LVVAL
      PRIOR2=(KURPRI*100)+NSNIOR
      IF(PRIOR2.GE.PRIOR1) GO TO 74
C     RESET PRIORITIES OF PROCESS IN MEMORY ALLOCATION QUEUE
 79   CONTINUE
C79   FILIST(+TABLE'INITPR,CURENT-.1 'INITPR')
      LVVAL=FILIST
      LVVARG=LVVAL
      LVV 1=LVVARG
      LVFUNC=TABLE
```

```
        CALL LVFIND
        INITPR
        1  =LVVAL
        LVVAL=LVV 1
        LVVARG=LVVAL
        LVFUNC=CURENT
        CALL LVFIND
        CALL LVFNV
        LVVALS(1)=INITPR
        LVTYPE(1)=1
        LVNDXN=1
        CALL LVNSRT
C       MEMORY QUEUE.K SINK/71/71
        LVVAL=MEMORY
        LVVARG=LVVAL
        LVFUNC=QUEUE
        CALL LVFIND
        LVVPOS=  K
        CALL LVFNV
        LVVALS(1)=SINK
        LVNDXN=2
        CALL LVNSRT
        IF(LVVTR .EQ. -1) GO TO    71
        IF(LVVTR .NE. -1) GO TO    71
C
C    ALL PROCESSES WHICH CAN BE SATISFIED HAVE BEEN GIVEN PERIPHERALS
C    TYPE OUT RESOURCE STATUS
C
C    UPDATE CURRENT PRIORITY OF PROCESSES STILL IN THE PERIPHERAL QUEUE
  72  I=0
 81     CONTINUE
C81   PERIPH+QUEUE.'I=I+1'/83+PROCES(+CURENT'KURPRI,CURENT-.1 'KURPRI+1'
C    + /81/81)
        LVVAL=PERIPH
        LVVARG=LVVAL
        LVFUNC=QUEUE
        CALL LVFIND
        I=I+1
        LVVPOS=  I
        CALL LVFNV
        IF(LVVTR .EQ. -1) GO TO    83
        LVVARG=LVVAL
        LVFUNC=PROCES
        CALL LVFIND
        LVV 1=LVVAL
        LVV 2=LVFUNC
        LVV 3=LVVARG
        LVVARG=LVVAL
        LVFUNC=CURENT
        CALL LVFIND
        KURPRI
        1  =LVVAL
        LVVAL=LVV 1
        LVVARG=LVVAL
        LVFUNC=CURENT
        CALL LVFIND
        CALL LVFNV
        LVVALS(1)=KURPRI+1
        LVTYPE(1)=1
        LVNDXN=1
        CALL LVNSRT
        IF(LVVTR .EQ. -1) GO TO    81
        IF(LVVTR .NE. -1) GO TO    81
C
C    TYPE OUT RESOURCE STATUS
  83  CALL DUMP
C
```

```
C
C     MEMORY IS ALLOCATED ON A FIRST FIT BASIS
C     SEARCH MEMORY QUEUE, PROCESSES ARE IN PRIORITY ORDER.
         I=0
 80      CONTINUE
C80    MEMORY+QUEUE.'I=I+1'/82'SINK+PROCES'SINK2+TABLE.2'MEMREQ
         LVVAL=MEMORY
         LVVARG=LVVAL
         LVFUNC=QUEUE
         CALL LVFIND
         I=I+1
         LVVPOS=  I
         CALL LVFNV
         IF(LVVTR .EQ. -1) GO TO    82
         SINK
         1  =LVVAL
         LVVARG=LVVAL
         LVFUNC=PROCES
         CALL LVFIND
         SINK2
         1  =LVVAL
         LVVARG=LVVAL
         LVFUNC=TABLE
         CALL LVFIND
         LVVPOS=      2
         CALL LVFNV
         MEMREQ
         1  =LVVAL
C
C     REJECT ANY PROCESS REQUESTING MEMORY IN EXCESS OF 100 WORDS.
         IF(MEMREQ.GT.100) CALL PURGE(MEMORY,SINK)
C
C     SEARCH MEMORY AVAILABLE LIST TO DETERMINE IF THE MEMORY
C     REQUIREMENTS OF THE PROCESS CAN BE MET.  IF SO, DELETE THAT MEMORY
C     BLOCK FROM THE MEMORY AVAILABLE LIST
         J=0
 86      CONTINUE
C86    MEMORY+(AVAIL(,'J=J+1'/80'MEMLFT<MEMREQ//86,-.J),QUEUE-,I)
         LVVAL=MEMORY
         LVVARG=LVVAL
         LVV 1=LVVARG
         LVFUNC=AVAIL
         CALL LVFIND
         LVV 2=LVVAL
         LVV 3=LVFUNC
         LVV 4=LVVARG
         J=J+1
         LVVPOS=  J
         CALL LVFNV
         IF(LVVTR .EQ. -1) GO TO    80
         MEMLFT
         1  =LVVAL
         LVVARG=LVVAL
         LVVTR=-1
         IF(LVVAL .LT. MEMREQ
         1 ) LVVTR=1
         IF(LVVTR .NE. -1) GO TO    86
         LVFUNC=LVV 3
         LVVARG=LVV 4
         CALL LVFIND
         LVVPOS=  J
         CALL LVFNV
         LVNDXN=1
         CALL LVDLET
         LVVAL=LVV 1
         LVVARG=LVVAL
```

```
      LVFUNC=QUEUE
      CALL LVFIND
      LVVPOS=  I
      CALL LVFNV
      LVNDXN=1
      CALL LVDLET
      I=I-1
C
C     A BLOCK OF MEMORY WHICH IS LARGE ENOUGH FOR THE PROCESS HAS BEEN
C     FOUND, MODIFY PROCESS TABLE SO THAT MEMORY REQUIREMENT BECOMES
C        SINK2 TABLE-.2 MEMLFT
      LVVAL=SINK2
      LVVARG=LVVAL
      LVFUNC=TABLE
      CALL LVFIND
      LVVPOS=      2
      CALL LVFNV
      LVVALS(1)=MEMLFT
      LVNDXN=1
      CALL LVNSRT
C
C*** INSERT PROCESS INTO READY LIST AND RESET ITS PRIORITY AND SENIORITY
C        SINK2(+TABLE'INITPR,CURENT-.1 'INITPR')
      LVVAL=SINK2
      LVVARG=LVVAL
      LVV 1=LVVARG
      LVFUNC=TABLE
      CALL LVFIND
      INITPR
      1   =LVVAL
      LVVAL=LVV 1
      LVVARG=LVVAL
      LVFUNC=CURENT
      CALL LVFIND
      CALL LVFNV
      LVVALS(1)=INITPR
      LVTYPE(1)=1
      LVNDXN=1
      CALL LVNSRT
C        READY QUEUE SINK SENIOR-.1 'SNRITY'/80/80
      LVVAL=READY
      LVVARG=LVVAL
      LVFUNC=QUEUE
      CALL LVFIND
      LVVALS(1)=SINK
      CALL LVNSRT
      LVVARG=LVVAL
      LVFUNC=SENIOR
      CALL LVFIND
      CALL LVFNV
      LVVALS(1)=SNRITY
      LVTYPE(1)=1
      LVNDXN=1
      CALL LVNSRT
      IF(LVVTR .EQ. -1) GO TO    80
      IF(LVVTR .NE. -1) GO TO    80
C
C     UPDATE PRIORITIES OF ALL PROCESSES LEFT IN THE MEMORY ALLOCATION
C     QUEUE
  82  M=0
 84     CONTINUE
C84    MEMORY+QUEUE.'M=M+1'/90+PROCES(+TABLE'INITPR,CURENT-.1'INITPR')
      LVVAL=MEMORY
      LVVARG=LVVAL
      LVFUNC=QUEUE
      CALL LVFIND
```

```
          M=M+1
          LVVPOS=  M
          CALL LVFNV
          IF(LVVTR .EQ. -1) GO TO    90
          LVVARG=LVVAL
          LVFUNC=PROCES
          CALL LVFIND
          LVV 1=LVVAL
          LVV 2=LVFUNC
          LVV 3=LVVARG
          LVVARG=LVVAL
          LVFUNC=TABLE
          CALL LVFIND
          INITPR
          1  =LVVAL
          LVVAL=LVV 1
          LVVARG=LVVAL
          LVFUNC=CURENT
          CALL LVFIND
          CALL LVFNV
          LVVALS(1)=INITPR
          LVTYPE(1)=1
          LVNDXN=1
          CALL LVNSRT
C
C*** SORT READY LIST
   90   I=0
          KSWICH=0
   91   I=I+1
C
C     COMPARE I'TH AND I+1'ST PRIORITY VALUES, SWITCH IF I+1'ST IS LARGER
C        READY+QUEUE.I+(PROCES+CURENT'KURPRI,SENIOR'NSNIOR)
          LVVAL=READY
          LVVARG=LVVAL
          LVFUNC=QUEUE
          CALL LVFIND
          LVVPOS=  I
          CALL LVFNV
          LVVARG=LVVAL
          LVV 1=LVVARG
          LVFUNC=PROCES
          CALL LVFIND
          LVVARG=LVVAL
          LVFUNC=CURENT
          CALL LVFIND
          KURPRI
          1  =LVVAL
          LVVAL=LVV 1
          LVVARG=LVVAL
          LVFUNC=SENIOR
          CALL LVFIND
          NSNIOR
          1  =LVVAL
          PRIOR1=(100*KURPRI)+NSNIOR
C         READY+QUEUE."I+1"/92'SINK+(PROCES+CURENT'KURPRI,SENIOR'NSNIOR)
          LVVAL=READY
          LVVARG=LVVAL
          LVFUNC=QUEUE
          CALL LVFIND
          LVVPOS=  I+1
          CALL LVFNV
          IF(LVVTR .EQ. -1) GO TO    92
          SINK
          1  =LVVAL
          LVVARG=LVVAL
          LVV 1=LVVARG
```

```
      LVFUNC=PROCES
      CALL LVFIND
      LVVARG=LVVAL
      LVFUNC=CURENT
      CALL LVFIND
      KURPRI
      1  =LVVAL
      LVVAL=LVV 1
      LVVARG=LVVAL
      LVFUNC=SENIOR
      CALL LVFIND
      NSNIOR
      1  =LVVAL
      PRIOR2=(100*KURPRI)+NSNIOR
      IF(PRIOR1.GE.PRIOR2) GO TO 91
C
C     SWITCH I'TH AND I+1'ST POSITIONS, SET SWITCH FLAG
      KSWICH=1
C     READY(+QUEUE-.'I+1',QUEUE.I SINK/91/91)
      LVVAL=READY
      LVVARG=LVVAL
      LVV 1=LVVARG
      LVFUNC=QUEUE
      CALL LVFIND
      LVVPOS=  I+1
      CALL LVFNV
      LVNDXN=1
      CALL LVDLET
      LVVAL=LVV 1
      LVVARG=LVVAL
      LVFUNC=QUEUE
      CALL LVFIND
      LVVPOS=  I
      CALL LVFNV
      LVVALS(1)=SINK
      LVNDXN=2
      CALL LVNSRT
      IF(LVVTR .EQ. -1) GO TO    91
      IF(LVVTR .NE. -1) GO TO    91
C
C     KEEP SORTING IF A CHANGE WAS MADE ON THE LAST PASS
 92   IF(KSWICH.EQ.1) GO TO 90
C
C     READY LIST HAS BEEN SORTED, REPORT OUT.
      CALL DUMP
C
C*** BEGIN CPU ALLOCATION ALGORITHM
C
C     IS A PROCESS EXECUTING?
 200  LEASTM=1000
      IF(CPURUN.NE.0) GO TO 204
C
C     TRANSFER HIGHEST PRIORITY PROCESS FROM READY LIST INTO CPU
C     READY+QUEUE/205'CPURUN -.1
      LVVAL=READY
      LVVARG=LVVAL
      LVFUNC=QUEUE
      CALL LVFIND
      IF(LVVTR .EQ. -1) GO TO   205
      CPURUN
      1  =LVVAL
      CALL LVFNV
      LVNDXN=1
      CALL LVDLET
C
C     UPDATE PRIORITIES OF ALL PROCESSES IN READY LIST
```

```
 204  I=0
 381    CONTINUE
C381  READY+QUEUE.'I=I+1'/205+PROCES(+CURENT'KURPRI,CURENT-.1'KURPRI+1'
C     + /381/381)
        LVVAL=READY
        LVVARG=LVVAL
        LVFUNC=QUEUE
        CALL LVFIND
        I=I+1
        LVVPOS=  I
        CALL LVFNV
        IF(LVVTR .EQ. -1) GO TO   205
        LVVARG=LVVAL
        LVFUNC=PROCES
        CALL LVFIND
        LVV 1=LVVAL
        LVV 2=LVFUNC
        LVV 3=LVVARG
        LVVARG=LVVAL
        LVFUNC=CURENT
        CALL LVFIND
        KURPRI
        1  =LVVAL
        LVVAL=LVV 1
        LVVARG=LVVAL
        LVFUNC=CURENT
        CALL LVFIND
        CALL LVFNV
        LVVALS(1)=KURPRI+1
        LVTYPE(1)=1
        LVNDXN=1
        CALL LVNSRT
        IF(LVVTR .EQ. -1) GO TO   381
        IF(LVVTR .NE. -1) GO TO   381
C
C     SEARCH BLOCKED-FOR-I/O LIST AND CPU FOR LEAST TIME
 205    CONTINUE
C205  CPURUN+PROCES/206+SEQVEN'LEASTM
        LVVAL=CPURUN
        LVVARG=LVVAL
        LVFUNC=PROCES
        CALL LVFIND
        IF(LVVTR .EQ. -1) GO TO   206
        LVVARG=LVVAL
        LVFUNC=SEQVEN
        CALL LVFIND
        LEASTM
        1  =LVVAL
  206 I=0
 210    CONTINUE
C210  IO+BLOKED.'I=I+1'/220+PROCES+SEQVEN'IOLSTM<LEASTM/210
        LVVAL=IO
        LVVARG=LVVAL
        LVFUNC=BLOKED
        CALL LVFIND
        I=I+1
        LVVPOS=  I
        CALL LVFNV
        IF(LVVTR .EQ. -1) GO TO   220
        LVVARG=LVVAL
        LVFUNC=PROCES
        CALL LVFIND
        LVVARG=LVVAL
        LVFUNC=SEQVEN
        CALL LVFIND
        IOLSTM
```

```
      1  =LVVAL
      LVVARG=LVVAL
      LVVTR=-1
      IF(LVVAL .LT. LEASTM
      1 ) LVVTR=1
      IF(LVVTR .EQ. -1) GO TO   210
      LEASTM=IOLSTM
      GO TO 210
C
C     TYPE OUT RESOURCE STATUS
  220 CALL DUMP
C
C     UPDATE TIME, SENIORITY
      TIME=TIME+LEASTM
      SNRITY=9999-TIME
C
C     REDUCE TIME OF CPU AND I/O PROCESSES BY LEASTM
C     CPURUN+PROCES/224(+SEQVEN'ITIME,SEQVEN-.1*ITIME-LEASTM*)
      LVVAL=CPURUN
      LVVARG=LVVAL
      LVFUNC=PROCES
      CALL LVFIND
      IF(LVVTR .EQ. -1) GO TO   224
      LVV 1=LVVAL
      LVV 2=LVFUNC
      LVV 3=LVVARG
      LVVARG=LVVAL
      LVFUNC=SEQVEN
      CALL LVFIND
      ITIME
      1  =LVVAL
      LVVAL=LVV 1
      LVVARG=LVVAL
      LVFUNC=SEQVEN
      CALL LVFIND
      CALL LVFNV
      LVVALS(1)=ITIME-LEASTM
      LVTYPE(1)=1
      LVNDXN=1
      CALL LVNSRT
  224 I=0
  225   CONTINUE
C225  IO+BLOKED.*I=I+1*/230+PROCES(+SEQVEN'ITIME,SEQVEN-.1*ITIME-LEASTM*
C    + /225/225)
      LVVAL=IO
      LVVARG=LVVAL
      LVFUNC=BLOKED
      CALL LVFIND
      I=I+1
      LVVPOS=  I
      CALL LVFNV
      IF(LVVTR .EQ. -1) GO TO   230
      LVVARG=LVVAL
      LVFUNC=PROCES
      CALL LVFIND
      LVV 1=LVVAL
      LVV 2=LVFUNC
      LVV 3=LVVARG
      LVVARG=LVVAL
      LVFUNC=SEQVEN
      CALL LVFIND
      ITIME
      1  =LVVAL
      LVVAL=LVV 1
      LVVARG=LVVAL
      LVFUNC=SEQVEN
```

```
        CALL LVFIND
        CALL LVFNV
        LVVALS(1)=ITIME-LEASTM
        LVTYPE(1)=1
        LVNDXN=1
        CALL LVNSRT
        IF(LVVTR .EQ. -1) GO TO   225
        IF(LVVTR .NE. -1) GO TO   225
C
C     HAS THE PROCESS IN EXECUTION FINISHED ITS TIME SEQUENCE
 230    CONTINUE
C230   CPURUN +PROCES/300+SEQVEN="0"/300
        LVVAL=CPURUN
        LVVARG=LVVAL
        LVFUNC=PROCES
        CALL LVFIND
        IF(LVVTR .EQ. -1) GO TO   300
        LVVARG=LVVAL
        LVFUNC=SEQVEN
        CALL LVFIND
        LVVTR=-1
        IF(LVVAL .EQ. 0
        1 ) LVVTR=1
        IF(LVVTR .EQ. -1) GO TO   300
C
C*** PROCESS IN CPU WILL SWITCH TO I/O UNLESS FINISHED OR BLOCKED
C
C     POP STACK OF SEQUENCE OF EVENTS
C       CPURUN+PROCES+SEQVEN-.1
        LVVAL=CPURUN
        LVVARG=LVVAL
        LVFUNC=PROCES
        CALL LVFIND
        LVVARG=LVVAL
        LVFUNC=SEQVEN
        CALL LVFIND
        CALL LVFNV
        LVNDXN=1
        CALL LVDLET
C
C     IS THIS PROCESS COMPLETELY FINISHED?
C       CPURUN+PROCES'SINK+SEQVEN.2'MEDIA=END//400
        LVVAL=CPURUN
        LVVARG=LVVAL
        LVFUNC=PROCES
        CALL LVFIND
        SINK
        1  =LVVAL
        LVVARG=LVVAL
        LVFUNC=SEQVEN
        CALL LVFIND
        LVVPOS=      2
        CALL LVFNV
        MEDIA
        1  =LVVAL
        LVVTR=-1
        IF(LVVAL .EQ. END
        1 ) LVVTR=1
        IF(LVVTR .NE. -1) GO TO   400
C
C     RESET TO CURRENT PRIORITY
C       SINK(+TABLE'INITPR,CURENT-.1 "INITPR")
        LVVAL=SINK
        LVVARG=LVVAL
        LVV 1=LVVARG
        LVFUNC=TABLE
```

```
        CALL LVFIND
        INITPR
        1  =LVVAL
        LVVAL=LVV 1
        LVVARG=LVVAL
        LVFUNC=CURENT
        CALL LVFIND
        CALL LVFNV
        LVVALS(1)=INITPR
        LVTYPE(1)=1
        LVNDXN=1
        CALL LVNSRT
        IF(MEDIA.EQ.OTHER) GO TO 240
C
C     NEXT SEQUENCE USES TAPE OR DISK, IF THE APPROPIATE CHANNEL IS BUSY,
C     PUT IN WAIT STATE
        IF(MEDIA.EQ.TAPE ) GO TO 250
C
C     SEARCH BLOCKED-DOING-I/O LIST FOR DISK
        I=0
 235    CONTINUE
C235   IO+BLOKED.'I=I+1'/240+PROCES+SEQVEN.2=DISK/235
        LVVAL=IO
        LVVARG=LVVAL
        LVFUNC=BLOKED
        CALL LVFIND
        I=I+1
        LVVPOS=  I
        CALL LVFNV
        IF(LVVTR .EQ. -1) GO TO   240
        LVVARG=LVVAL
        LVFUNC=PROCES
        CALL LVFIND
        LVVARG=LVVAL
        LVFUNC=SEQVEN
        CALL LVFIND
        LVVPOS=      2
        CALL LVFNV
        LVVTR=-1
        IF(LVVAL .EQ. DISK
        1 ) LVVTR=1
        IF(LVVTR .EQ. -1) GO TO   235
C
C     DISK IS ON BLOCKED-DOING-I/O LIST, PLACE ON WAITING-FOR-DISK QUEUE
C       IO WAITDK CPURUN/260/260
        LVVAL=IO
        LVVARG=LVVAL
        LVFUNC=WAITDK
        CALL LVFIND
        LVVALS(1)=CPURUN
        CALL LVNSRT
        IF(LVVTR .EQ. -1) GO TO   260
        IF(LVVTR .NE. -1) GO TO   260
C
C     SEARCH BLOCKED-DOING-I/O LIST FOR TAPE
  250 I=0
 255    CONTINUE
C255   IO+BLOKED.'I=I+1'/240+PROCES+SEQVEN.2=TAPE/255
        LVVAL=IO
        LVVARG=LVVAL
        LVFUNC=BLOKED
        CALL LVFIND
        I=I+1
        LVVPOS=  I
        CALL LVFNV
        IF(LVVTR .EQ. -1) GO TO   240
```

```
        LVVARG=LVVAL
        LVFUNC=PROCES
        CALL LVFIND
        LVVARG=LVVAL
        LVFUNC=SEQVEN
        CALL LVFIND
        LVVPOS=      2
        CALL LVFNV
        LVVTR=-1
        IF(LVVAL .EQ. TAPE
        1 ) LVVTR=1
        IF(LVVTR .EQ. -1) GO TO   255
C
C     TAPE IS ON BLOCKED-DOING-I/O LIST, PLACE ON WAITING-FOR-TAPE QUEUE
C       IO WAITTP CPURUN/260/260
        LVVAL=IO
        LVVARG=LVVAL
        LVFUNC=WAITTP
        CALL LVFIND
        LVVALS(1)=CPURUN
        CALL LVNSRT
        IF(LVVTR .EQ. -1) GO TO   260
        IF(LVVTR .NE. -1) GO TO   260
C
C     PLACE MEDIA ON BLOCKED-DOING-I/O LIST
 240    CONTINUE
C240   IO BLOKED CPURUN
        LVVAL=IO
        LVVARG=LVVAL
        LVFUNC=BLOKED
        CALL LVFIND
        LVVALS(1)=CPURUN
        CALL LVNSRT
C
C     PLACE HIGHEST PRIORITY PROCESS INTO EXECUTION
 260    CONTINUE
C260   READY+QUEUE /290'CPURUN -.1//300
        LVVAL=READY
        LVVARG=LVVAL
        LVFUNC=QUEUE
        CALL LVFIND
        IF(LVVTR .EQ. -1) GO TO   290
        CPURUN
        1  =LVVAL
        CALL LVFNV
        LVNDXN=1
        CALL LVDLET
        IF(LVVTR .NE. -1) GO TO   300
C
C       READY LIST IS EMPTY, NO JOB IS RUNNING
  290 CPURUN=0
C
C*** SEARCH BLOCKED-DOING-I/O LIST FOR COMPLETED I/O
  300 I=0
 310    CONTINUE
C310   IO+BLOKED.'I=I+1'/350'SINK+PROCES'SINK2+SEQVEN='0'/310
        LVVAL=IO
        LVVARG=LVVAL
        LVFUNC=BLOKED
        CALL LVFIND
        I=I+1
        LVVPOS=  I
        CALL LVFNV
        IF(LVVTR .EQ. -1) GO TO   350
        SINK
        1  =LVVAL
```

```
      LVVARG=LVVAL
      LVFUNC=PROCES
      CALL LVFIND
      SINK2
      1  =LVVAL
      LVVARG=LVVAL
      LVFUNC=SEQVEN
      CALL LVFIND
      LVVTR=-1
      IF(LVVAL .EQ. 0
      1 ) LVVTR=1
      IF(LVVTR .EQ. -1) GO TO   310
C
C*** PROCESS IS TO BE TAKEN OFF THE I/O LIST AND PUT ON THE READY LIST
C     UPDATE SENIORITY
C        SINK SENIOR-.1 'SNRITY'
         LVVAL=SINK
         LVVARG=LVVAL
         LVFUNC=SENIOR
         CALL LVFIND
         CALL LVFNV
         LVVALS(1)=SNRITY
         LVTYPE(1)=1
         LVNDXN=1
         CALL LVNSRT
C
C        RESET CURRENT PRIORITY TO INITIAL PRIORITY
C        SINK2(+TABLE'INITPR,CURENT-.1 'INITPR')
         LVVAL=SINK2
         LVVARG=LVVAL
         LVV 1=LVVARG
         LVFUNC=TABLE
         CALL LVFIND
         INITPR
         1  =LVVAL
         LVVAL=LVV 1
         LVVARG=LVVAL
         LVFUNC=CURENT
         CALL LVFIND
         CALL LVFNV
         LVVALS(1)=INITPR
         LVTYPE(1)=1
         LVNDXN=1
         CALL LVNSRT
         PRIOR1=(INITPR*100)+SNRITY
C
C     PLACE PROCESS INTO READY LIST IN POSITION WRT PRIORITY
         J=0
 320     CONTINUE
C320   READY+QUEUE.'J=J+1'/340+(SENIOR'NSNIOR,PROCES+CURENT'KURPRI)
         LVVAL=READY
         LVVARG=LVVAL
         LVFUNC=QUEUE
         CALL LVFIND
         J=J+1
         LVVPOS=  J
         CALL LVFNV
         IF(LVVTR .EQ. -1) GO TO   340
         LVVARG=LVVAL
         LVV 1=LVVARG
         LVFUNC=SENIOR
         CALL LVFIND
         NSNIOR
         1  =LVVAL
         LVVAL=LVV 1
         LVVARG=LVVAL
```

```
          LVFUNC=PROCES
          CALL LVFIND
          LVVARG=LVVAL
          LVFUNC=CURENT
          CALL LVFIND
          KURPRI
          1  =LVVAL
          PRIOR2=(KURPRI*100)+NSNIOR
          IF(PRIOR2.GE.PRIOR1) GO TO 320
  340     CONTINUE
C340   READY QUEUE.J SINK
          LVVAL=READY
          LVVARG=LVVAL
          LVFUNC=QUEUE
          CALL LVFIND
          LVVPOS=  J
          CALL LVFNV
          LVVALS(1)=SINK
          LVNDXN=2
          CALL LVNSRT
C
C     REMOVE FROM BLOCKED-DOING I/O LIST
C         IO+BLOKED-.I
          LVVAL=IO
          LVVARG=LVVAL
          LVFUNC=BLOKED
          CALL LVFIND
          LVVPOS=  I
          CALL LVFNV
          LVNDXN=1
          CALL LVDLET
          I=I-1
C
C     REMOVE MEDIA AND I/O TIME FROM SEQUENCE OF EVENTS
C         SINK2 +SEQVEN-.(1,1'MEDIA=OTHER//310)
          LVVAL=SINK2
          LVVARG=LVVAL
          LVFUNC=SEQVEN
          CALL LVFIND
          LVV 1=LVVAL
          LVV 2=LVFUNC
          LVV 3=LVVARG
          CALL LVFNV
          LVNDXN=1
          CALL LVDLET
          LVFUNC=LVV 2
          LVVARG=LVV 3
          CALL LVFIND
          CALL LVFNV
          LVNDXN=1
          CALL LVDLET
          MEDIA
          1  =LVVAL
          LVVTR=-1
          IF(LVVAL .EQ. OTHER
          1 ) LVVTR=1
          IF(LVVTR .NE. -1) GO TO   310
C
C     CHECK I/O WAITING LISTS AND TRANSFER TO I/O-BLOCKED LIST
          IF(MEDIA.EQ.TAPE) GO TO 345
C         IO(+WAITDK/310'SINK3-.1,BLOKED SINK3/310/310)
          LVVAL=IO
          LVVARG=LVVAL
          LVV 1=LVVARG
          LVFUNC=WAITDK
          CALL LVFIND
```

```
      IF(LVVTR .EQ. -1) GO TO   310
      SINK3
      1  =LVVAL
      CALL LVFNV
      LVNDXN=1
      CALL LVDLET
      LVVAL=LVV 1
      LVVARG=LVVAL
      LVFUNC=BLOKED
      CALL LVFIND
      LVVALS(1)=SINK3
      CALL LVNSRT
      IF(LVVTR .EQ. -1) GO TO   310
      IF(LVVTR .NE. -1) GO TO   310
 345  CONTINUE
C345  IO(+WAITTP/310'SINK3-.1,BLOKED SINK3/310/310)
      LVVAL=IO
      LVVARG=LVVAL
      LVV 1=LVVARG
      LVFUNC=WAITTP
      CALL LVFIND
      IF(LVVTR .EQ. -1) GO TO   310
      SINK3
      1  =LVVAL
      CALL LVFNV
      LVNDXN=1
      CALL LVDLET
      LVVAL=LVV 1
      LVVARG=LVVAL
      LVFUNC=BLOKED
      CALL LVFIND
      LVVALS(1)=SINK3
      CALL LVNSRT
      IF(LVVTR .EQ. -1) GO TO   310
      IF(LVVTR .NE. -1) GO TO   310
C
C     COMPARE PRIORITIES OF PROCESS WITH FIRST PROCESS IN READY LIST
 350  CONTINUE
C350  CPURUN+PROCES/200+CURENT'KURPR1
      LVVAL=CPURUN
      LVVARG=LVVAL
      LVFUNC=PROCES
      CALL LVFIND
      IF(LVVTR .EQ. -1) GO TO   200
      LVVARG=LVVAL
      LVFUNC=CURENT
      CALL LVFIND
      KURPR1
      1  =LVVAL
C     READY+QUEUE/200+PROCES+CURENT'KURPR2
      LVVAL=READY
      LVVARG=LVVAL
      LVFUNC=QUEUE
      CALL LVFIND
      IF(LVVTR .EQ. -1).GO TO   200
      LVVARG=LVVAL
      LVFUNC=PROCES
      CALL LVFIND
      LVVARG=LVVAL
      LVFUNC=CURENT
      CALL LVFIND
      KURPR2
      1  =LVVAL
      IF(KURPR1.GE.KURPR2) GO TO 200
C
C    PRIORITY OF PROCESS IN READY LIST IS HIGHER THAN CPU, SWITCH.
```

```
      KTEMP=CPURUN
C     READY+QUEUE'CPURUN-.1
      LVVAL=READY
      LVVARG=LVVAL
      LVFUNC=QUEUE
      CALL LVFIND
      CPURUN
      1  =LVVAL
      CALL LVFNV
      LVNDXN=1
      CALL LVDLET
C
C     PUT DISPLACED PROCESS INTO READY LIST IN POSITION WRT PRIORITY
      PRIOR1=KURPR1*100
      I=0
 370  CONTINUE
C370  READY+QUEUE.'I=I+1'/375+(SENIOR'NSNIOR,PROCES+CURENT'KURPRI)
      LVVAL=READY
      LVVARG=LVVAL
      LVFUNC=QUEUE
      CALL LVFIND
      I=I+1
      LVVPOS=  I
      CALL LVFNV
      IF(LVVTR .EQ. -1) GO TO   375
      LVVARG=LVVAL
      LVV 1=LVVARG
      LVFUNC=SENIOR
      CALL LVFIND
      NSNIOR
      1  =LVVAL
      LVVAL=LVV 1
      LVVARG=LVVAL
      LVFUNC=PROCES
      CALL LVFIND
      LVVARG=LVVAL
      LVFUNC=CURENT
      CALL LVFIND
      KURPRI
      1  =LVVAL
      PRIOR2=(KURPRI*100)+NSNIOR
      IF(PRIOR2.GE.PRIOR1) GO TO 370
 375  CONTINUE
C375  READY QUEUE.I KTEMP
      LVVAL=READY
      LVVARG=LVVAL
      LVFUNC=QUEUE
      CALL LVFIND
      LVVPOS=  I
      CALL LVFNV
      LVVALS(1)=KTEMP
      LVNDXN=2
      CALL LVNSRT
C
C     UPDATE PRIORITY AND SENIORITY OF OLD CPU PROCESS
C        KTEMP(SENIOR-.1 'SNRITY',+PROCES(+TABLE'INITPR,
C     + CURENT-.1 'INITPR-1'))
        LVVAL=KTEMP
        LVVARG=LVVAL
        LVV 1=LVVARG
        LVFUNC=SENIOR
        CALL LVFIND
        CALL LVFNV
        LVVALS(1)=SNRITY
        LVTYPE(1)=1
        LVNDXN=1
```

```
        CALL LVNSRT
        LVVAL=LVV 1
        LVVARG=LVVAL
        LVFUNC=PROCES
        CALL LVFIND
        LVV 2=LVVAL
        LVV 3=LVFUNC
        LVV 4=LVVARG
        LVVARG=LVVAL
        LVFUNC=TABLE
        CALL LVFIND
        INITPR
        1  =LVVAL
        LVVAL=LVV 2
        LVVARG=LVVAL
        LVFUNC=CURENT
        CALL LVFIND
        CALL LVFNV
        LVVALS(1)=INITPR-1
        LVTYPE(1)=1
        LVNDXN=1
        CALL LVNSRT
        GO TO 200
C
C*** PROCESS IS FINISHED, REALLOCATE RESOURCES, DELETE THAT PROCESS FROM
C    INPUT QUEUE, INSERT NEXT PROCESS OF THAT JOB INTO THE PERIPHERAL
C    ALLOCATION QUEUE.
C
C    TYPE NAME OF PROCESS WHICH IS FINISHED
 400    CONTINUE
C400  CPURUN+(NAME'JOBNAM,PROCES+TABLE.3'NPROC)
        LVVAL=CPURUN
        LVVARG=LVVAL
        LVV 1=LVVARG
        LVFUNC=NAME
        CALL LVFIND
        JOBNAM
        1  =LVVAL
        LVVAL=LVV 1
        LVVARG=LVVAL
        LVFUNC=PROCES
        CALL LVFIND
        LVVARG=LVVAL
        LVFUNC=TABLE
        CALL LVFIND
        LVVPOS=      3
        CALL LVFNV
        NPROC
        1  =LVVAL
        TYPE 10, TIME,JOBNAM,NPROC
  10  FORMAT(///,5X,'TIME  ',I3,A2,'.'I1,' IS FINISHED')
C
C    RETURN MEMORY
C       CPURUN+PROCES'NODE+TABLE.2'MEMREQ
        LVVAL=CPURUN
        LVVARG=LVVAL
        LVFUNC=PROCES
        CALL LVFIND
        NODE
        1  =LVVAL
        LVVARG=LVVAL
        LVFUNC=TABLE
        CALL LVFIND
        LVVPOS=      2
        CALL LVFNV
        MEMREQ
```

```
        1  =LVVAL
C       MEMORY AVAIL 'MEMREQ'
        LVVAL=MEMORY
        LVVARG=LVVAL
        LVFUNC=AVAIL
        CALL LVFIND
        LVVALS(1)=MEMREQ
        LVTYPE(1)=1
        CALL LVNSRT
C
C       RETURN PERIPHERAL DEVICES
        I=0
 410    CONTINUE
C410  NODE+FILREQ.'I=I+1'/420'NFILE
        LVVAL=NODE
        LVVARG=LVVAL
        LVFUNC=FILREQ
        CALL LVFIND
        I=I+1
        LVVPOS=  I
        CALL LVFNV
        IF(LVVTR .EQ. -1) GO TO   420
        NFILE
        1  =LVVAL
C       PERIPH AVAIL _NFILE/410/410
        LVVAL=PERIPH
        LVVARG=LVVAL
        LVFUNC=AVAIL
        CALL LVFIND
        LVTYPE(1)=3
        LVVALS(1)=NFILE
        CALL LVNSRT
        IF(LVVTR .EQ. -1) GO TO   410
        IF(LVVTR .NE. -1) GO TO   410
C
C    REMOVE PROCESS FROM INPUT QUEUE
 420    CONTINUE
C420  CPURUN+PROCES-.1-(CURENT,TABLE,SEQVEN,FILREQ)
        LVVAL=CPURUN
        LVVARG=LVVAL
        LVFUNC=PROCES
        CALL LVFIND
        CALL LVFNV
        LVNDXN=1
        CALL LVDLET
        LVVARG=LVVAL
        LVV 1=LVVARG
        LVFUNC=CURENT
        CALL LVDLET
        LVVAL=LVV 1
        LVVARG=LVVAL
        LVFUNC=TABLE
        CALL LVDLET
        LVVAL=LVV 1
        LVVARG=LVVAL
        LVFUNC=SEQVEN
        CALL LVDLET
        LVVAL=LVV 1
        LVVARG=LVVAL
        LVFUNC=FILREQ
        CALL LVDLET
C
C    REMOVE PROCESS FROM CPU
        KTEMP=CPURUN
        CPURUN=0
C
```

```
C    DOES THIS JOB HAVE ANY MORE PROCESSES?
C       KTEMP+PROCES/500
        LVVAL=KTEMP
        LVVARG=LVVAL
        LVFUNC=PROCES
        CALL LVFIND
        IF(LVVTR .EQ. -1) GO TO   500
C
C*** PLACE THE NEXT PROCESS OF THIS JOB INTO THE PERIPHERAL
C    ALLOCATION QUEUE ACCORDING TO PRIORITY
C       KTEMP+PROCES+CURENT'KURPRI
        LVVAL=KTEMP
        LVVARG=LVVAL
        LVFUNC=PROCES
        CALL LVFIND
        LVVARG=LVVAL
        LVFUNC=CURENT
        CALL LVFIND
        KURPRI
        1  =LVVAL
        PRIOR1=(KURPRI*100)+SNRITY
        I=0
 430    CONTINUE
C430  PERIPH+QUEUE.'I=I+1'/440+(SENIOR'NSNIOR,PROCES+CURENT'KURPRI)
        LVVAL=PERIPH
        LVVARG=LVVAL
        LVFUNC=QUEUE
        CALL LVFIND
        I=I+1
        LVVPOS=  I
        CALL LVFNV
        IF(LVVTR .EQ. -1) GO TO   440
        LVVARG=LVVAL
        LVV 1=LVVARG
        LVFUNC=SENIOR
        CALL LVFIND
        NSNIOR
        1  =LVVAL
        LVVAL=LVV 1
        LVVARG=LVVAL
        LVFUNC=PROCES
        CALL LVFIND
        LVVARG=LVVAL
        LVFUNC=CURENT
        CALL LVFIND
        KURPRI
        1  =LVVAL
        PRIOR2=(KURPRI*100)+NSNIOR
        IF(PRIOR2.GE.PRIOR1) GO TO 430
 440    CONTINUE
C440  PERIPH QUEUE.I KTEMP/70/70
        LVVAL=PERIPH
        LVVARG=LVVAL
        LVFUNC=QUEUE
        CALL LVFIND
        LVVPOS=  I
        CALL LVFNV
        LVVALS(1)=KTEMP
        LVNDXN=2
        CALL LVNSRT
        IF(LVVTR .EQ. -1) GO TO    70
        IF(LVVTR .NE. -1) GO TO    70
C
C*** THIS JOB HAS NO MORE PROCESSES
 500    CONTINUE
C500  INPUT+QUEUE-.: KTEMP
```

```
      LVVAL=INPUT
      LVVARG=LVVAL
      LVFUNC=QUEUE
      CALL LVFIND
      LVVINC=KTEMP
      CALL LVINCL
      CALL LVFNV
      LVNDXN=1
      CALL LVDLET
C
C     IF THERE ARE PROCESSES LEFT IN THE PERIPHERAL ALLOCATION QUEUE,
C                                                          GO TO 70
C       PERIPH+QUEUE//70
      LVVAL=PERIPH
      LVVARG=LVVAL
      LVFUNC=QUEUE
      CALL LVFIND
      IF(LVVTR .NE. -1) GO TO    70
C
C     IF THERE ARE PROCESSES LEFT IN THE MEMORY ALLOCATION QUEUE,
C                                                          GO TO 83
C       MEMORY+QUEUE//83
      LVVAL=MEMORY
      LVVARG=LVVAL
      LVFUNC=QUEUE
      CALL LVFIND
      IF(LVVTR .NE. -1) GO TO    83
C
C     IF THERE ARE PROCESSES LEFT IN THE READY QUEUE, GO TO 200
C       READY +QUEUE//200
      LVVAL=READY
      LVVARG=LVVAL
      LVFUNC=QUEUE
      CALL LVFIND
      IF(LVVTR .NE. -1) GO TO   200
      STOP
      END
```

```
      SUBROUTINE DUMP
      IMPLICIT INTEGER(A-Z)
      COMMON /LVARGS/ LVFUNC,LVVARG,LVVPOS,LVVTYP,LVVAL,
     1 LVVNVL,LVSKIP,LVVTR,LVVINC,LVNDXN,LVVALS(10),LVTYPE(10)
      COMMON QUEUE,NAME,SENIOR,PROCES,CURENT,TABLE,SEQVEN,IO,CPURUN,
     1 FILREQ,TIME,AVAIL,PERIPH,MEMORY,READY,BLOKED,WAITDK,WAITTP
      DIMENSION IFILES(12)
      DATA BLANK/1H /
C
C     THIS ROUTINE IS AN EXECUTIVE FOR TYPEING OUT ALL SYSTEM PARTICULARS
C
      TYPE 1,TIME
    1 FORMAT(///,5X,'TIME     ',I5,//)
C
C     TYPE PERIPHERALS AVAILABLE
      DO 20 I=1,12
   20 IFILES(I)=BLANK
      TYPE 2
    2 FORMAT(//,6X,'PERIPHERALS AVAILABLE')
      I=0
 10   CONTINUE
C10   PERIPH+AVAIL.*I=I+1*/25 'IPERIP
      LVVAL=PERIPH
      LVVARG=LVVAL
      LVFUNC=AVAIL
      CALL LVFIND
      I=I+1
      LVVPOS=  I
      CALL LVFNV
      IF(LVVTR .EQ. -1) GO TO    25
      IPERIP
      1  =LVVAL
      IFILES(I)=IPERIP
      GO TO 10
   25 TYPE 3,(IFILES(I),I=1,12)
    3 FORMAT(6X,12(A2,1X)///)
C
C     TYPE PERIPHERAL ALLOCATION QUEUE
      TYPE 4
    4 FORMAT(6X,'PERIPHERAL ALLOCATION QUEUE')
C
C     SEARCH PERIPHERAL ALLOCATION QUEUE FOR PROCESS PARTICULARS- JOB NAM
C     CURRENT PRIORITY, AND PROCESS NUMBER.
      CALL REPORT(PERIPH)
C
C     TYPE AVAILABLE MEMORY
      DO 90 I=1,4
   90 IFILES(I)=0
      I=0
 91   CONTINUE
C91   MEMORY+AVAIL.*I=I+1*/95'MEM
      LVVAL=MEMORY
      LVVARG=LVVAL
      LVFUNC=AVAIL
      CALL LVFIND
      I=I+1
      LVVPOS=  I
      CALL LVFNV
      IF(LVVTR .EQ. -1) GO TO    95
      MEM
      1  =LVVAL
      IFILES(I)=MEM
      GO TO 91
   95 TYPE 6,(IFILES(I),I=1,4)
    6 FORMAT(//,6X,'MEMORY AVAILABLE      ',4I5,//)
C
C     TYPE MEMORY ALLOCATION QUEUE
      TYPE 7
    7 FORMAT(6X,'MEMORY ALLOCATION QUEUE')
```

```
C     SEARCH MEMORY ALLOCATION QUEUE FOR PROCESS PARTICULARS
         CALL REPORT(MEMORY)
C
C     TYPE READY LIST
         TYPE 8
    8   FORMAT(//,6X,'READY LIST')
         M=0
   16    CONTINUE
C16     READY+QUEUE.*M=M+1*/17+(NAME'JOBNAM,PROCES+(CURENT'KURPRI,
C     + SEQVEN'TIMLFT,TABLE.3'NPROC))
         LVVAL=READY
         LVVARG=LVVAL
         LVFUNC=QUEUE
         CALL LVFIND
         M=M+1
         LVVPOS=  M
         CALL LVFNV
         IF(LVVTR .EQ. -1) GO TO    17
         LVVARG=LVVAL
         LVV 1=LVVARG
         LVFUNC=NAME
         CALL LVFIND
         JOBNAM
         1  =LVVAL
         LVVAL=LVV 1
         LVVARG=LVVAL
         LVFUNC=PROCES
         CALL LVFIND
         LVVARG=LVVAL
         LVV 2=LVVARG
         LVFUNC=CURENT
         CALL LVFIND
         KURPRI
         1  =LVVAL
         LVVAL=LVV 2
         LVVARG=LVVAL
         LVFUNC=SEQVEN
         CALL LVFIND
         TIMLFT
         1  =LVVAL
         LVVAL=LVV 2
         LVVARG=LVVAL
         LVFUNC=TABLE
         CALL LVFIND
         LVVPOS=       3
         CALL LVFNV
         NPROC
         1  =LVVAL
         TYPE 11,JOBNAM,NPROC,KURPRI,TIMLFT
         GO TO 16
C
C     TYPE STATUS OF PROCESS BEING EXECUTED
   17   TYPE 9
    9   FORMAT(//,6X,'PROCESS IN EXECUTION')
         IF(CPURUN.GT.0) GO TO 30
C     NO PROCESS IS RUNNING
         TYPE 15
   15   FORMAT(8X,'NONE',//)
         GO TO 40
C
C     CPURUN POINTS TO PROCESS WHICH IS RUNNING
   30    CONTINUE
C30     CPURUN+(SENIOR'NSNRTY,NAME'JOBNAM,PROCES+(CURENT'KURPRI,
C     + SEQVEN'TIMLFT,TABLE.3'NPROC))
         LVVAL=CPURUN
         LVVARG=LVVAL
         LVV 1=LVVARG
         LVFUNC=SENIOR
         CALL LVFIND
```

```
      NSNRTY
      1  =LVVAL
      LVVAL=LVV 1
      LVVARG=LVVAL
      LVFUNC=NAME
      CALL LVFIND
      JOBNAM
      1  =LVVAL
      LVVAL=LVV 1
      LVVARG=LVVAL
      LVFUNC=PROCES
      CALL LVFIND
      LVVARG=LVVAL
      LVV 2=LVVARG
      LVFUNC=CURENT
      CALL LVFIND
      KURPRI
      1  =LVVAL
      LVVAL=LVV 2
      LVVARG=LVVAL
      LVFUNC=SEQVEN
      CALL LVFIND
      TIMLFT
      1  =LVVAL
      LVVAL=LVV 2
      LVVARG=LVVAL
      LVFUNC=TABLE
      CALL LVFIND
      LVVPOS=      3
      CALL LVFNV
      NPROC
      1  =LVVAL
      TYPE 11,JOBNAM,NPROC,KURPRI,TIMLFT
 11   FORMAT(6X,A2,'.',I1,2X,'PRIORITY ',I2,4X,'CPU TIME ',I2,//)
C
C     TYPE BLOCKED-DOING-I/O LIST
 40   TYPE 12
 12   FORMAT(6X,'BLOCKED-DOING-I/O LIST')
        CALL IORPRT(BLOKED)
C
C     TYPE DISK I/O QUEUE
        TYPE 13
 13   FORMAT(//,6X,'LIST OF JOBS BLOCKED, WAITING FOR DISK CHANNEL')
        CALL IORPRT(WAITDK)
C
C     TYPE TAPE I/O QUEUE
        TYPE 14
 14   FORMAT(//,6X,'LIST OF JOBS BLOCKED, WAITING FOR TAPE CHANNEL')
        CALL IORPRT(WAITTP)
        RETURN
        END
```

```
      SUBROUTINE REPORT(NODE)
      IMPLICIT INTEGER(A-Z)
      COMMON /LVARGS/ LVFUNC,LVVARG,LVVPOS,LVVTYP,LVVAL,
      1 LVVNVL,LVSKIP,LVVTR,LVVINC,LVNDXN,LVVALS(10),LVTYPE(10)
      COMMON QUEUE,NAME,SENIOR,PROCES,CURENT,TABLE,SEQVEN,IO,CPURUN,
      1 FILREQ
C
C     THIS ROUTINE SEARCHES THE QUEUE NAMED BY NODE AND REPORTS OUT
C
      I=0
 10   CONTINUE
C10   NODE+QUEUE/18.*I=I+1*/15+(SENIOR'NSNRTY,NAME'JOBNAM,PROCES+
C     + (CURENT'KURPRI,TABLE.3'NPROC))
      LVVAL=NODE
      LVVARG=LVVAL
      LVFUNC=QUEUE
      CALL LVFIND
      IF(LVVTR .EQ. -1) GO TO    18
      I=I+1
      LVVPOS=  I
      CALL LVFNV
      IF(LVVTR .EQ. -1) GO TO    15
      LVVARG=LVVAL
      LVV 1=LVVARG
      LVFUNC=SENIOR
      CALL LVFIND
      NSNRTY
      1  =LVVAL
      LVVAL=LVV 1
      LVVARG=LVVAL
      LVFUNC=NAME
      CALL LVFIND
      JOBNAM
      1  =LVVAL
      LVVAL=LVV 1
      LVVARG=LVVAL
      LVFUNC=PROCES
      CALL LVFIND
      LVVARG=LVVAL
      LVV 2=LVVARG
      LVFUNC=CURENT
      CALL LVFIND
      KURPRI
      1  =LVVAL
      LVVAL=LVV 2
      LVVARG=LVVAL
      LVFUNC=TABLE
      CALL LVFIND
      LVVPOS=      3
      CALL LVFNV
      NPROC
      1  =LVVAL
      TYPE 5, JOBNAM,NPROC,KURPRI
 5    FORMAT(6X,A2,'.',I1,2X,'PRIORITY ',I2,/)
      GO TO 10
 18   TYPE 17
 17   FORMAT(8X,'NONE',//)
 15   CONTINUE
      RETURN
      END
```

```
      SUBROUTINE IORPRT(LINK)
      IMPLICIT INTEGER(A-Z)
      COMMON /LVARGS/ LVFUNC,LVVARG,LVVPOS,LVVTYP,LVVAL,
      1 LVVNVL,LVSKIP,LVVTR,LVVINC,LVNDXN,LVVALS(10),LVTYPE(10)
      COMMON QUEUE,NAME,SENIOR,PROCES,CURENT,TABLE,SEQVEN,IO,CPURUN
      COMMON/IOFILE/ DISK,TAPE,OTHER,END
      DATA ITAPE,IEND,IDISK,IOTHER/2HTP,2HEN,2HDK,2HOT/
C
C     THIS ROUTINE SEARCHES AND REPORTS ON THE REQUESTED I/O QUEUE
C
      I=0
 10   CONTINUE
C10   IO+LINK/18.*I=I+1*/15+(SENIOR'NSNRTY,NAME'JOBNAM,PROCES+
C     + (CURENT'KURPRI,TABLE.3'NPROC,SEQVEN('TIMLFT,.2'MEDIA)))
      LVVAL=IO
      LVVARG=LVVAL
      LVFUNC=LINK
      CALL LVFIND
      IF(LVVTR .EQ. -1) GO TO    18
      I=I+1
      LVVPOS=  1
      CALL LVFNV
      IF(LVVTR .EQ. -1) GO TO    15
      LVVARG=LVVAL
      LVV 1=LVVARG
      LVFUNC=SENIOR
      CALL LVFIND
      NSNRTY
      1  =LVVAL
      LVVAL=LVV 1
      LVVARG=LVVAL
      LVFUNC=NAME
      CALL LVFIND
      JOBNAM
      1  =LVVAL
      LVVAL=LVV 1
      LVVARG=LVVAL
      LVFUNC=PROCES
      CALL LVFIND
      LVVARG=LVVAL
      LVV 2=LVVARG
      LVFUNC=CURENT
      CALL LVFIND
      KURPRI
      1  =LVVAL
      LVVAL=LVV 2
      LVVARG=LVVAL
      LVFUNC=TABLE
      CALL LVFIND
      LVVPOS=      3
      CALL LVFNV
      NPROC
      1  =LVVAL
      LVVAL=LVV 2
      LVVARG=LVVAL
      LVFUNC=SEQVEN
      CALL LVFIND
      LVV 3=LVVAL
      LVV 4=LVFUNC
      LVV 5=LVVARG
      TIMLFT
      1  =LVVAL
      LVVAL=LVV 3
      LVVPOS=      2
      CALL LVFNV
      MEDIA
      1  =LVVAL
```

```
      IF(MEDIA.EQ.OTHER) MED  =IOTHER
      IF(MEDIA.EQ.TAPE ) MED  =ITAPE
      IF(MEDIA.EQ.DISK ) MED  =IDISK
      IF(MEDIA.EQ.END  ) MED  =IEND
      TYPE 12, JOBNAM,NPROC,KURPRI,TIMLFT,MED
      GO TO 10
12    FORMAT(6X,A2,'.',I1,2X,'PRIORITY ',I2,4X,'I/O TIME LEFT ',I2,4X,
     1 A2,///)
18    TYPE 17
17    FORMAT(8X,'NONE',//)
15    CONTINUE
      RETURN
      END
```

```
      SUBROUTINE PURGE(IFILE,NODE)
      IMPLICIT INTEGER(A-Z)
      COMMON /LVARGS/ LVFUNC,LVVARG,LVVPOS,LVVTYP,LVVAL,
      1 LVVNVL,LVSKIP,LVVTR,LVVINC,LVNDXN,LVVALS(10),LVTYPE(10)
      COMMON QUEUE,NAME,SENIOR,PROCES,CURENT,TABLE,SEQVEN,IO,CPURUN,
      1 FILREQ
C
C     DELETE FROM THE GRAPH THE OLDEST PROCESS OF THE JOB
C       NODE+PROCES-.1-(CURENT,TABLE,FILREQ,SEQVEN)
        LVVAL=NODE
        LVVARG=LVVAL
        LVFUNC=PROCES
        CALL LVFIND
        CALL LVFNV
        LVNDXN=1
        CALL LVDLET
        LVVARG=LVVAL
        LVV 1=LVVARG
        LVFUNC=CURENT
        CALL LVDLET
        LVVAL=LVV 1
        LVVARG=LVVAL
        LVFUNC=TABLE
        CALL LVDLET
        LVVAL=LVV 1
        LVVARG=LVVAL
        LVFUNC=FILREQ
        CALL LVDLET
        LVVAL=LVV 1
        LVVARG=LVVAL
        LVFUNC=SEQVEN
        CALL LVDLET
C
C     WAS THAT THE LAST PROCESS OF THE JOB?
C       NODE+PROCES//RETURN
        LVVAL=NODE
        LVVARG=LVVAL
        LVFUNC=PROCES
        CALL LVFIND
        IF(LVVTR .NE. -1) RETURN
C
C     DELETE REMAINING INFORMATION ABOUT THE JOB (GARBAGE COLLECTION)
C       NODE-(SENIOR,NAME)
        LVVAL=NODE
        LVVARG=LVVAL
        LVV 1=LVVARG
        LVFUNC=SENIOR
        CALL LVDLET
        LVVAL=LVV 1
        LVVARG=LVVAL
        LVFUNC=NAME
        CALL LVDLET
C
C     DELETE FROM SPECIFIED QUEUE
C       IFILE+QUEUE-.: CPURUN
        LVVAL=IFILE
        LVVARG=LVVAL
        LVFUNC=QUEUE
        CALL LVFIND
        LVVINC=CPURUN
        CALL LVINCL
        CALL LVFNV
        LVNDXN=1
        CALL LVDLET
        RETURN
        END
```

APPENDIX B

EXECUTION TIMES OF THE BASIC GIRS OPERATIONS

The average times required to perform the various GIRS operations are dependent on the average length of the conflict lists, and in some cases, on the length of the multivalued list involved. Note that the average length of the conflict lists will rise as the GIRS buffer fills up, representing a traditional time-space tradeoff. Furthermore, since the length of multivalued lists is a function of program design, the programmer who must use the long lists should consider using the saved-index option as described in the Section "Retrieval".

The following table is based on one taken from Berkowitz[2] and modified to indicate insertion, deletion, and retrieval times for the unpacked and packed versions of GIRS on the CDC 6700 computing system. The MVL increment describes the extra time needed to access the "next" value of an MVL. The table also indicates the time required to traverse one function of a conflict list.

AVERAGE CDC-6600 EXECUTION TIMES FOR GIRS ROUTINES
(in microseconds)

	Unpacked Version		Packed Version	
	Minimum	MVL Increment	Minimum	MVL Increment
INSERT	60	0	185	0
FIND	50	15	76	28
DELETE	63	28	192	169
TRAVERSE CONFLICT LIST		23		66

On the PDP-11, a single retrieval requires 678 microseconds with an MVL increment of 157 microseconds.*

*Data supplied by James R. Carlberg of the Computer Sciences and Information Systems Division.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C

GIRS SUBROUTINES USED ONLY WITH THE GIRL PREPROCESSOR

The two routines discussed here are to be used only when the FORTRAN calls to the GIRS routines have been created by the GIRL preprocessor. The purpose of these routines is to stack and pop internal GIRS variables for nested GIRL statements, thereby sidestepping a great deal of code creation. The degree of nesting allowed is to be determined by the user, although a default has been set at five sets of parentheses.

Subroutine LVSTACK

Function:

Stores pertinent variables from COMMON/LVARGS/ when a left parenthesis is found in a GIRL statement.

Calling Format:

CALL LVSTAK

Input Parameters:

(In COMMON/LVSTAK/)

MAXLEV Number of levels of parenthesis desired. Default is five.

Abstract:

In this routine, the following variables from COMMON/LVARGS/ are stacked:

IFUNC, IARG, IPOS, ITYP, IVAL, ITESTR, INCLUD, INDXON, IVALS(1), ITYP1(1)

Program Length:

CDC 6700		PDP-11
Unpacked	Packed	
64_8 (52)	64_8 (52)	274_8 (188)

Subroutine LVPOP

Function:

Outputs pertinent variables from COMMON/LVARGS/ when a comma is encountered in a GIRL statement.

Calling Format:

CALL LVPOP

Abstract:

In this routine, the following variables from COMMON/LVARGS/ are returned from the stack:

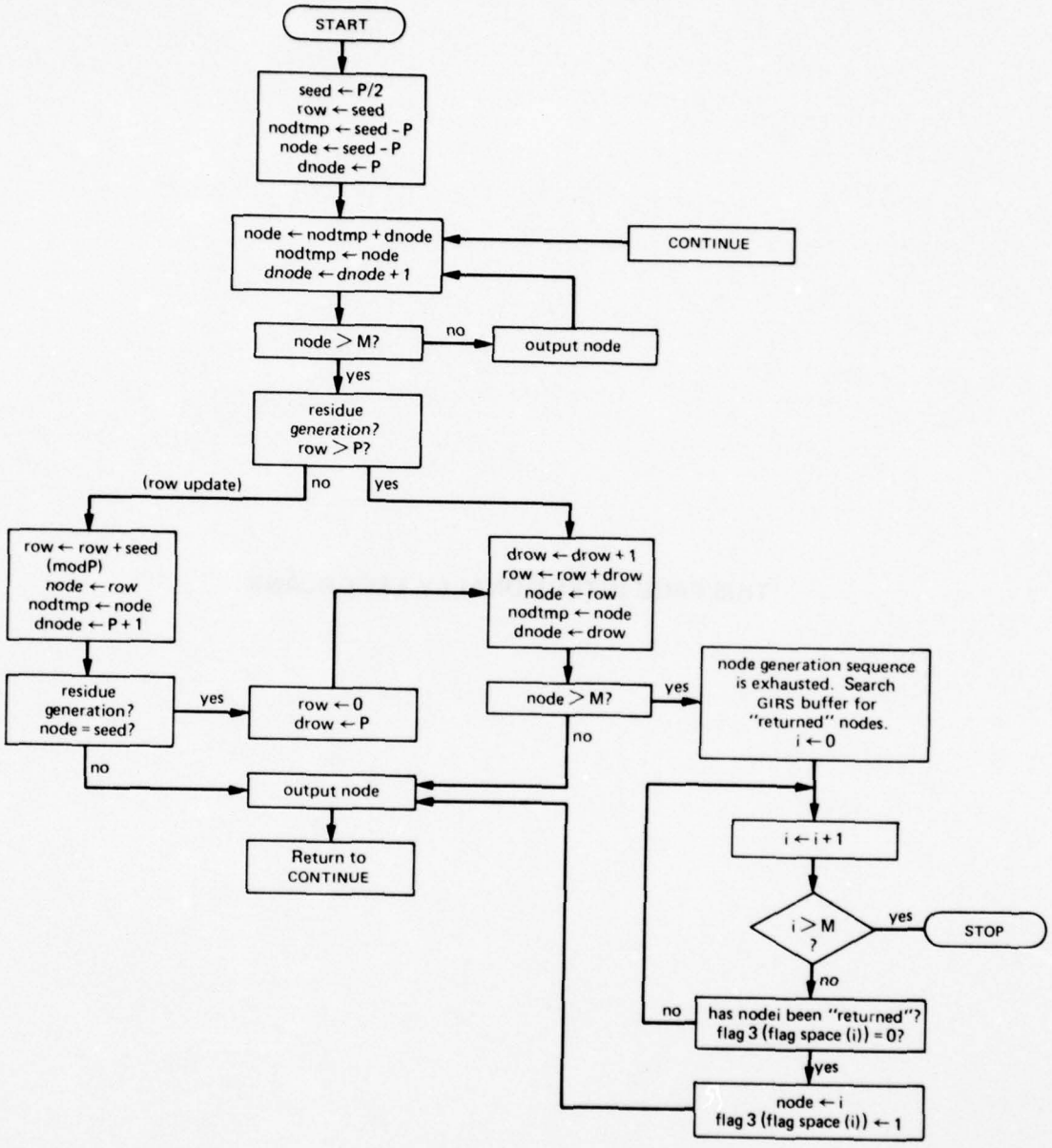
IFUNC, IARG, IPOS, ITYP, IVAL, ITESTR, INCLUD, INDXON, IVALS(1), ITYP1(1)

Program Length:

CDC 6700		PDP 11
Unpacked	Packed	
37_8 (31)	37_8 (31)	158_8 (111)

APPENDIX D

ALGORITHM FOR GENERATING THE SEQUENCE OF RANDOM NUMBERS



Note:

This algorithm is a modified version of the one which appears on page 22 of Berkowitz' Report 3531.[1]

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E

MEMORY REQUIREMENTS

The memory requirements for the labeled commons, GIRS, and the GIRL preprocessors are indicated in Table 4. The length in words of each subroutine, as used on the CDC 6700 and on the PDP-11, is indicated in Table 5.

TABLE 4 - MEMORY REQUIREMENTS FOR GIRS AND FOR THE GIRL PREPROCESSOR

	CDC 6700				PDP-11	
	Unpacked		Packed			
	Decimal	Octal	Decimal	Octal	Decimal	Octal
Main Memory: (in words)						
Total GIRS routines	1492	2724	2128	4120	4923	11473
Subset of GIRS for the pre-processor	976	172			3619	7043
GIRS labeled common	74	112	74	112	64	100
Preprocessor routines	4053	7725			12082	27462
Preprocessor labeled common	3524	6704			3202	6202
Preprocessor buffer length	840	1510				
Preprocessor total	9467	22373			18967	45027
Disk Space: (in PRU's* and blocks**)						
GIRS (object code)	51 PRU's		56 PRU's		162 blocks** (OBJ file)	
Preprocessor (object code)	81 PRU's				91 blocks (SAV file)	

*64 words per PRU.
**256 words per block.

TABLE 5 - LENGTHS OF GIRS SUBROUTINES
(in words)

Subroutines	CDC 6700				PDP-11	
	Unpacked		Packed			
	Decimal	Octal	Decimal	Octal	Decimal	Octal
LVSETP	42	52	63	77	146	222
LVGRN	48	60	48	60	167	247
LVFIND	33	41	59	73	215	327
LVFNV	159	237	176	260	453	705
LVINCL	33	41	33	41	109	155
LVNSRT	389	605	729	1331	1670	3206
LVUPDT	7	7	28	34	37	45
LVDLET	95	137	263	407	472	730
LVDUMP	170	252	161	241	389	605
LVFECH	79	117	52	64	157	235
LVCMPN	242	362	246	366	806	1446
LVUNPK	71	107				
LVPACK			146	222		
LVRTSH	16	20	16	20		
LVLFSH	25	31	25	31		
LVPOP	31	37	31	37	111	157
LVSTAK	52	64	52	64	188	274
Total	1492	2724	2128	4120	4923	11473

APPENDIX F

VARIABLES IN LABELED COMMON

The labeled common blocks used by GIRS are as follows:

For all implementations:

```
/LVARGS/ IFUNC,IARG,IPOS,ITYP,IVAL,NVAL,NSKIP,ITESTR,INCLUD,
    [INDXON,]* IVALS(10),ITYP(10)
/LVADDR/ IADD,THIS,LSTHED,LOC,LAST
/LVRAND/ KPRIME,KSEED,NROW,KDNODE,KDROW,KTEMP
/LVTABL/ MAPSZE,IEXTRA,MAP(mapsze or 1)**
/LVVSEQ/ ISEQSZ,ISQPOS,LASTSQ,SEQSPC(iseqsz or 1)
/LVFLAG/ FL0MSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,FLG67
/LVCRNT/ REGASP
/LVBUFR/ MEMSZE,BINFIL,KOMPAN
```

For the PDP-11 and the CDC unpacked version of GIRS:

```
/LVVTR1/ NODSPC (memsze)
/LVVTR2/ LSTSPC (memsze)
/LVVTR3/ LNKSPC (memsze)
/LVVTR4/ FLGSPC (memsze)
/LVVTR5/ NODESP (old memory size or 1)**
/LVVTR6/ LISTSP (old memory size or 1)
/LVVTR7/ LINKSP (old memory size or 1)
/LVVTR8/ FLAGSP (old memory size or 1)
```

The Packed Version of GIRS (CDC):

```
/LVVTR1/ WRKSPC (memsze)
/LVVTR5/ WORKSP (old memory size or 1)
```

---

*PDP-11

**Set MAPSZE and "old memory size" to 1, unless the buffer which contains the graph is to be compressed or expanded as described under initialization and in the section on using subroutine LVCMPN. The routines /LVFLAG/, /LVMASK/, and /LVSHFT/ are not needed in the user's calling routine.

PRECEDING PAGE BLANK-NOT FILMED

For either the packed or unpacked version of GIRS (CDC):

```
/LVMASK/ MASK1,MASK2,MASK3,MASK4,[NMASK1,NMASK2,NMASK3,]* NMASK4
/LVSHFT/ ISHFT1,ISHFT2,ISHFT3
```

---

*Packed version only.

APPENDIX G

SUBROUTINE LISTINGS

CDC 6000 Implementation - Unpacked Version

```
      SUBROUTINE LVSETP
      INTEGER FLGSPC,FLAGSP,BINFIL,SEQSPC,REGASP
      COMMON/LVVTR1/MEMSZE,REGASP,NODSPC(    1)/LVVTR2/LSTSPC(    1)/
     *LVVTR3/LNKSPC(    1)/LVVTR4/FLGSPC(    1)
      COMMON /LVVSEQ/ISEQSZ,ISQPOS,LASTSQ,SEQSPC(1)
      COMMON/LVRAND/ KPRIME,KSEED,NROW,KDNODE,KDROW,KTEMP
      COMMON/LVVTR5/BINFIL,KOMPAN,NODESP(1)/LVVTR6/LISTSP(1)
     + /LVVTR7/LINKSP(1)/LVVTR8/FLAGSP(1)
      IF(KOMPAN.NE.0) CALL LVCMPN
      KSEED=KPRIME/2
      NROW=KSEED
      KTEMP=KSEED-KPRIME
      KDNODE=KPRIME
      CALL LVGRN(REGASP)
      IOLD=REGASP
      DO 10 I=2,MEMSZE
      LNKSPC(I)=0
      FLGSPC(I)=0
      CALL LVGRN(NEW)
      NODSPC(NEW)=IOLD
      LSTSPC(IOLD)=NEW
   10 IOLD=NEW
      FLGSPC(1)=0
      LNKSPC(1)=0
      NODSPC(REGASP)=IOLD
      LSTSPC(IOLD)=REGASP
      NROW=KSEED
      KTEMP=KSEED-KPRIME
      KDNODE=KPRIME
      IF(KOMPAN.NE.0) CALL LVCMPN
      RETURN
      END
```

```
      SUBROUTINE LVFECH(N)
      INTEGER FLGSPC,SEQSPC,REGASP
      COMMON/LVVTR1/MEMSZE,REGASP,NODSPC(    1)/LVVTR2/LSTSPC(    1)/
     *LVVTR3/LNKSPC(    1)/LVVTR4/FLGSPC(    1)
      COMMON /LVVSEQ/ISEQSZ,ISQPOS,LASTSQ,SEQSPC(1)
      COMMON/LVRAND/ KPRIME,KSEED,NROW,KDNODE,KDROW,KTEMP
      READ(N) REGASP,MEMSZE,KPRIME,KSEED,NROW,KDNODE,KTEMP,KDROW,
     + ISEQSZ,MAPSZE
      READ(N)(NODSPC(I),I=1,MEMSZE)
      READ(N)(LSTSPC(I),I=1,MEMSZE)
      READ(N)(LNKSPC(I),I=1,MEMSZE)
      READ(N)(FLGSPC(I),I=1,MEMSZE)
      READ(N)(SEQSPC(I),I=1,ISEQSZ)
      PRINT 10
   10 FORMAT(1H ,*  GRAPH HAS BEEN PLACED INTO MEMORY*,//)
      RETURN
      END
```

```
      SUBROUTINE LVGRN(NODE)
      INTEGER FLGSPC,REGASP
      COMMON/LVVTR1/MEMSZE,REGASP,NODSPC(    1)/LVVTR2/LSTSPC(    1)/
     *LVVTR3/LNKSPC(    1)/LVVTR4/FLGSPC(    1)
      COMMON/LVRAND/ KPRIME,KSEED,NROW,KDNODE,KDROW,KTEMP
      NODE=KTEMP+KDNODE
      KTEMP=NODE
      KDNODE=KDNODE+1
      IF(NODE.GT.MEMSZE) GO TO 5
      RETURN
C     RESIDUE GENERATION ?
    5 IF(NROW.GT.KPRIME) GO TO 10
      NROW=NROW+KSEED
      IF(NROW.GT.KPRIME) NROW=NROW-KPRIME
      NODE=NROW
      KTEMP=NODE
      KDNODE=KPRIME+1
C     RESIDUE GENERATION ?
      IF(NODE.NE.KSEED) RETURN
      NROW=0
      KDROW=KPRIME
C     RESIDUE GENERATION
   10 KDROW=KDROW+1
      NROW=NROW+KDROW
      NODE=NROW
      KTEMP=NODE
      KDNODE=KDROW
      IF(NODE.GT.MEMSZE) GO TO 20
      RETURN
   20 PRINT 15
   15 FORMAT(1H ,*ERROR...NUMBER OF NODES EXCEEDS REQUESTED MEMORY.*/*
     *     PROGRAM IS TERMINATED.*)
      STOP
      END
```

```
      SUBROUTINE LVFIND
      INTEGER FLGSPC,REGASP,FL0MSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,
     + FLG67,SEQSPC,THIS
      COMMON /LVARGS/IFUNC,IARG,IPOS,ITYP,IVAL,NVAL,NSKIP,ITESTR,INCLUD,
     + IVALS(10),ITYP1(10)
      COMMON/LVVTR1/MEMSZE,REGASP,NODSPC(   1)/LVVTR2/LSTSPC(   1)/
     *LVVTR3/LNKSPC(   1)/LVVTR4/FLGSPC(   1)
      COMMON /LVVSEQ/ISEQSZ,ISQPOS,LASTSQ,SEQSPC(1)
      COMMON /LVADDR/ IADD,THIS,LSTHED,LOC,LAST
      COMMON/LVFLAG/FL0MSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,FLG67
      DATA FL0MSK/200B/,FL1MSK/100B/,FL2MSK/40B/,FL5MSK/4B/,FLG67/3B/,
     + FL3MSK/20B/,FL4MSK/10B/
C
C     IADD   = COMPUTED FUNCTION ADDRESS
C     THIS   = LOCATION OF FUNCTION ON CONFLICT LIST
C     LOC    = LOCATION OF RETRIEVED VALUE
C     LSTHED = -1, SINGLE VALUED LIST
C            = 0, NO LIST IS FOUND
C            = .GT.0, ADDRESS OF FIRST VALUE
C     ITESTR = 1, RETRIEVAL IS SUCCESSFUL (IVAL = RETURNED VALUE)
C            = -1, RETRIEVAL IS FAILURE (IVAL = SOURCE NODE)
      ITESTR=1
      IADD=IFUNC+IARG
      IF(IADD.GT.MEMSZE) IADD=IADD-MEMSZE
      LSTHED=0
      THIS=IADD
      IF((FLGSPC(THIS).AND.FL5MSK).EQ.0) GO TO 99
C     SEARCH CONFLICT LIST FOR KEY (IFUNC OR LINK)
    1 IF(NODSPC(THIS).EQ.IFUNC) GO TO 4
      LAST=THIS
      THIS=LNKSPC(THIS)
      IF((FLGSPC(THIS).AND.FL5MSK).NE.0) GO TO 99
      GO TO 1
C
C     THE FUNCTION HAS BEEN FOUND.
C     TEST FOR SINGLE VALUE LIST (SVL)  OR MULTIVALUED LIST (MVL).
    4 IF((FLGSPC(THIS).AND.FL0MSK).NE.0)  GO TO 14
C     SINGLE VALUED LIST.
      LSTHED=-1
      LOC=THIS
      IVAL=LSTSPC(LOC)
      GO TO 5
C     MULTIVALUED LIST.  OBTAIN FIRST VALUE.
   14 LSTHED=LSTSPC(THIS)
      LOC=LSTHED
      IVAL=NODSPC(LOC)
      GO TO 5
   99 ITESTR=-1
      IVAL=IARG
C     RESET TO DEFAULT VALUES
    5 IPOS=1
      ITYP=3
      RETURN
      END
```

```
      SUBROUTINE LVFNV(INDEXS)
      INTEGER FLGSPC,REGASP,FL0MSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,
     + FLG67,SEQSPC,THIS
      COMMON/LVVTR1/MEMSZE,REGASP,NODSPC(   1)/LVVTR2/LSTSPC(   1)/
     *LVVTR3/LNKSPC(   1)/LVVTR4/FLGSPC(   1)
      COMMON /LVVSEQ/ISEQSZ,ISQPOS,LASTSQ,SEQSPC(1)
      COMMON /LVADDR/ IADD,THIS,LSTHED,LOC,LAST
      COMMON /LVARGS/IFUNC,IARG,IPOS,ITYP,IVAL,NVAL,NSKIP,ITESTR,INCLUD,
     + IVALS(10),ITYP1(10)
      COMMON/LVFLAG/FL0MSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,FLG67
      COMMON /LVMASK/ MASK1,MASK2,MASK3,MASK4,NMASK4
      COMMON /LVSHFT/ ISHFT1,ISHFT2,ISHFT3
      DATA MASK1/777777000000000000008/,MASK2/777777000000008/,
     + MASK3/77777000B/,MASK4/777B/,NMASK4/7777777777777777770008/
      DATA ISHFT1,ISHFT2,ISHFT3/42,24,9/
      DATA NFLAG4/3678/
C
C     LVFIND MUST BE CALLED IMMEDIATELY PRIOR TO THE CALL TO THIS ROUTINE
C     INPUT IS EXPECTED THRU COMMONS LVARGS AND LVADDR.  THIS ROUTINE
C     SEARCHES THE MULTIVALUE LIST FOR THE IPOS'TH VALUE OF THE REQUESTED
C     TYPE.  IF SVL, TYPE MUST BE EITHER UNSPECIFIED OR CORRECT.
C
C     DOES THE FUNCTION EXIST ?
      IF(ITESTR.LT.0) GO TO 70
      JPOS=IABS(IPOS)
      IF(LSTHED.GT.0) GO TO 60
C     SVL -  DOES FUNCTION QUALIFY ?
      IF(JPOS.NE.1) GO TO 99
      IF(ITYP.EQ.3) GO TO 70
      ISTYP=(FLGSPC( LOC).AND.FLG67)
      IF(ISTYP.EQ.3)ISTYP=2
      IF(ISTYP.NE.ITYP)  GO TO 99
      GO TO 70
C
C     MVL - FIRST VALUE HAS ALREADY BEEN FOUND BY LVFIND, SET INDEX
C     PARAMETERS.
   60 IF(IPOS .EQ. 1 .AND. ITYP .EQ. 3) GO TO 75
C
C     BEGIN SEARCH
      IND=0
C     IF THE SAVED INDEX FACILITY IS NOT TO BE USED, GO TO 50
      IF(NSKIP.EQ.1) GO TO 50
      IF(INDEXS.EQ.0) GO TO 50
C
C     UNPACK THE INPUT PARAMETERS FOR SAVED INDEX
C     INDEX = POSITION FROM TOP OR BOTTOM OF PREVIOUSLY RETRIEVED VALUE
C     INDXAD= LOCATION OF PREVIOUSLY RETRIEVED VALUE
C     KFUNC = FUNCTION CONTAINING PREVIOUSLY RETRIEVED VALUE
C     KARG  = ARGUMENT CONTAINING PREVIOUSLY RETRIEVED VALUE
      KARG  = LVRTSH((INDEXS.AND.MASK1),ISHFT1)
      KFUNC = LVRTSH((INDEXS.AND.MASK2),ISHFT2)
      INDXAD= LVRTSH((INDEXS.AND.MASK3),ISHFT3)
      INDEX =         (INDEXS.AND.MASK4)
C     IS THE INDEX NEGATIVE ?
      IF(INDEX.GE.256) INDEX=INDEX.OR.NMASK4
C
C     SAVED INDEX CAN'T BE USED IF IMMEDIATE PAST HISTORY = INDEXED
```

```
C     INSERTION OR DELETION.
      IF((FLGSPC(THIS) .AND. FL4MSK) .NE. 0) GO TO 50
C
C     SAVED INDEX CAN'T BE USED IF SOURCE NODE OR LINK HAVE BEEN CHANGED
      IF((KFUNC.NE.IFUNC).OR.(KARG.NE.IARG)) GO TO 50
C
C     SAVED INDEX CAN'T BE USED IF DIRECTION OF SEARCH HAS SWITCHED
      IF((IPOS*INDEX).LE.0) GO TO 50
      NDX=FLGSPC(INDXAD)
C
C     SAVED INDEX CAN'T BE USED IF VALUE AT SAVED INDEX HAS BEEN MOVED
      IF((NDX.AND.FL5MSK).NE.0)  GO TO 50
C
C     SAVED INDEX CAN'T BE USED IF VALUE AT SAVED INDEX HAS BEEN REMOVED
      IF((NDX.AND.FL1MSK).EQ.0)  GO TO 50
C
C     IS SEARCH FROM BEGINNING FASTER THAN FROM SAVED INDEX ?
      KNDEX=IABS(INDEX)
      IF(JPOS.LT.2) GO TO 50
      IF((JPOS+JPOS).LE.KNDEX)  GO TO 50
C
C     SAVED INDEX CAN BE USED, BEGIN SEARCH AT INDXAD.
      LOC=INDXAD
C     FIND RELATIVE DISTANCE FROM SAVED INDEX AND DETERMINE WHETHER TO
C     COUNT UP OR DOWN.  IF REQUESTED POSITION IS CLOSER TO THE BEGINNING
C     OF THE LIST THAN THE SAVED INDEX, COUNT UP, OTHERWISE, COUNT DOWN.
C
      LENGTH=INDEX-IPOS
      JPOS=IABS(LENGTH)
      IF(LENGTH) 10,28,40
C
COUNT UP FROM INDXADD
C
  40  ITOP=0
      GO TO 23
C
C     DO NOT USE SAVED INDEX.  START FROM THE BEGINNING OR END OF LIST
C
  50  FLGSPC(THIS)=FLGSPC(THIS).AND.NFLAG4
      IF(IPOS) 20,99,12
C
COUNT DOWN
C
  10  LOC=LSTSPC(LOC)
      IF((FLGSPC(LOC).AND.FL0MSK).NE.0)  GO TO 99
  12  IF(ITYP.EQ.3) GO TO 22
      ISTYP=(FLGSPC(LOC).AND.FLG67)
      IF(ISTYP.EQ.3)ISTYP=2
      IF(ISTYP.NE.ITYP) GO TO 10
  22  IND=IND+1
      IF(IND.NE.JPOS) GO TO 10
      GO TO 28
C
COUNT UP FROM THE BOTTOM OF THE LIST
C
  20  ITOP=1
  23  LOC=LNKSPC(LOC)
```

```
      IF(ITOP.EQ.1) GO TO 24
      IF((FLGSPC(LSTSPC(LOC)).AND.FLOMSK).NE.0)  GO TO 99
   24 ITOP=0
      IF(ITYP.EQ.3) GO TO 21
      ISTYP=FLGSPC(LOC).AND.FLG67
      IF(ISTYP.EQ.3)ISTYP=2
      IF(ISTYP.NE.ITYP) GO TO 23
   21 IND=IND+1
      IF(IND.NE.JPOS) GO TO 23
   28 IVAL=NODSPC(LOC)
C
C     SAVE INDEX PARAMETERS AFTER SUCCESSFUL RETRIEVAL
C
   75 IF(NSKIP.EQ.1) GO TO 70
      KARG  =LVLFSH( IARG,ISHFT1)
      KFUNC =LVLFSH(IFUNC,ISHFT2)
      INDXAD=LVLFSH(LOC,ISHFT3)
      INDEX=IPOS.AND.MASK4
      INDEXS=KARG.OR.KFUNC.OR.INDXAD.OR.INDEX
      GO TO 70
C
C    FAILURE EXIT
   99 ITESTR=-1
      IF(NSKIP.NE.1) INDEXS=0
      IVAL=IARG
C
C    SUCCESS EXIT, SET DEFAULTS.
   70 ITYP=3
      RETURN
      END
```

```
      SUBROUTINE LVINCL
      INTEGER THIS
      COMMON /LVARGS/IFUNC,IARG,IPOS,ITYP,IVAL,NVAL,NSKIP,ITESTR,INCLUD,
     + IVALS(10),ITYP1(10)
      COMMON /LVADDR/ IADD,THIS,LSTHED,LOC,LAST
C
C     THIS ROUTINE SEARCHES THE LIST TO FIND THE VALUE IN INCLUD.  IF IT
C     IS FOUND, ITS POSITION WRT THE TOP OF THE LIST IS RETURNED
C     DOES THE LIST EXIST ?
      IF(ITESTR.LT.0) GO TO 31
      IF(IVAL.EQ.INCLUD) GO TO 25
      IF(LSTHED.LT.0) GO TO 31
C     MVL FCUNO
      JVAL=IVAL
      KSKIP=NSKIP
      NSKIP=0
      INDEX=0
      KPOS=0
   10 KPOS=KPOS+1
      IPOS=KPOS
      CALL LVFNV(INDEX)
      IF(ITESTR.LT.0) GO TO 30
      IF(IVAL.NE.INCLUD) GO TO 10
C
C     EXIT FROM LOOP,  ITESTR= 1, SUCCESS
C                      ITESTR=-1, FAILURE
C     EXCEPT FOR IPOS, OUTPUT MUST APPEAR AS IF LVFIND WAS CALLED
   30 NSKIP=KSKIP
      IVAL=JVAL
      LOC=LSTHED
   25 INCLUD=ITESTR
      ITESTR=1
      RETURN
   31 INCLUD=-1
      RETURN
      END
```

```
      SUBROUTINE LVNSRT
      INTEGER FLGSPC,REGASP,FL0MSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,
     + FLG67,SEQSPC,THIS
     +,FLGTMP,TWO,THREE,HEAD,OLDLOC,ASPREG,TEMP,SVLRPL
      COMMON/LVVTR1/MEMSZE,REGASP,NODSPC(    1)/LVVTR2/LSTSPC(    1)/
     *LVVTR3/LNKSPC(    1)/LVVTR4/FLGSPC(    1)
      COMMON /LVADDR/ IADD,THIS,LSTHED,LOC,LAST
      COMMON /LVVSEQ/ISEQSZ,ISQPOS,LASTSQ,SEQSPC(1)
      COMMON /LVARGS/IFUNC,IARG,IPOS,ITYP,IVAL,NVAL,NSKIP,ITESTR,INCLUD,
     + IVALS(10),ITYP1(10)
      COMMON/LVFLAG/FL0MSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,FLG67
      DATA TWO/2B/,THREE/3B/,NFLG67/374B/,SVLRPL/0/
C
C     CALLS TO LVFIND OR LVFNV MUST PRECEDE A CALL TO THIS ROUTINE.
C
C     IS THE GIRS BUFFER FULL ?
 125   IF(REGASP.EQ.LSTSPC(REGASP)) GO TO 98
C
C    FORM FIRST WORD OF SINGLE OR MULTIVALUED FUNCTION
       FLGTMP=FL1MSK.OR.ITYP1(1)
       IF(NVAL.EQ.1)GO TO 20
       LSTTMP=REGASP
       FLGTMP=FLGTMP.OR.FL0MSK.OR.FL2MSK
       GO TO 21
  20   LSTTMP=IVALS(1)
C
C     IF THIS FUNCTION ALREADY EXISTS, GO TO 43
  21   IF(ITESTR.GT.0) GO TO 43
C
C    IF THAT ADDRESS IS ALREADY IN WORKING SPACE, GO TO 25
       IF((FL1MSK.AND.FLGSPC(IADD)).NE.0) GO TO 25
C
C     UPDATE REGASP(IF NECESSARY)
       IF(IADD.EQ.REGASP)REGASP=LSTSPC(IADD)
C
C    UPDATE AVAILABLE SPACE
       LSTSPC(NODSPC(IADD))=LSTSPC(IADD)
       NODSPC(LSTSPC(IADD))=NODSPC(IADD)
C
C     INSERT FUNCTION
       NODSPC(IADD)=IFUNC
       LSTSPC(IADD)=LSTTMP
       LNKSPC(IADD)=IADD
       FLGSPC(IADD)=FLGSPC(IADD).OR.FLGTMP.OR.FL5MSK
C
C    INSERT ANY ADDITIONAL FUNCTION VALUES
       IF(NVAL.EQ.1) GO TO 100
       HEAD=IADD
       OLDLOC=IADD
       GO TO 50
C
C     IF THAT ADDRESS CONTAINS THE HEAD OF A CONFLICT LIST, GO TO 60
  25   IF((FL5MSK.AND.FLGSPC(IADD)).GT.0) GO TO 60
C
C    IF THAT ADDRESS CONTAINS A VALUE ON A MULTIVALUE LIST, GO TO 35
       IF((FL2MSK.AND.FLGSPC(IADD)).GT.0.AND.(FL0MSK.AND.FLGSPC(IADD)).EQ
      *.0) GO TO 35
```

```
C
C-------------------------------------------------------------------------
C-THE ADDRESS CONTAINS A FUNCTION ON A CONFLICT LIST,BUT NOT THE HEAD OF
      THIS=IADD
C
C   FIND THE PRECEDING FUNCTION ON THE CONFLICT LIST
 26   IF(LNKSPC(LNKSPC(THIS)).EQ.IADD)GO TO 27
      THIS=LNKSPC(THIS)
      GO TO 26
 27   LAST=LNKSPC(THIS)
      NEWLOC=REGASP
      IF(REGASP.EQ.LSTSPC(REGASP)) GO TO 98
C
C   UPDATE AVAILABLE SPACE AND REGASP
      CALL LVUPDT

C   MOVE THE FUNCTION ON A CONFLICT LIST TO THE FIRST CELL OF AVAILABLE
C    SPACE
      NODSPC(NEWLOC)=NODSPC(IADD)
      LSTSPC(NEWLOC)=LSTSPC(IADD)
      LNKSPC(NEWLOC)=LNKSPC(IADD)
      FLGSPC(NEWLOC)=FLGSPC(IADD).OR.FL4MSK
      FLGSPC(IADD)=0
      LNKSPC(LAST)=NEWLOC
C
C   INSERT THIS FUNCTION AS THE HEAD OF A CONFLICT LIST
      NODSPC(IADD)=IFUNC
      LNKSPC(IADD)=IADD
      LSTSPC(IADD)=LSTTMP
      FLGSPC(IADD)=FLGSPC(IADD).OR.FLGTMP.OR.FL4MSK.OR.FL5MSK
      IF((FLGSPC(NEWLOC).AND.FL0MSK).EQ.0)  GO TO 34
C
C   IF THE FUNCTION THAT WAS MOVED IS THE HEAD OF A MULTIVALUE LIST,FIX
      NEXT=LSTSPC(NEWLOC)
 30   NEXT=LSTSPC(NEXT)
      IF(LSTSPC(NEXT).NE.IADD)GO TO 30
      LSTSPC(NEXT)=NEWLOC
C
C   INSERT ANY ADDITIONAL FUNCTION VALUES
 34   HEAD=IADD
      OLDLOC=IADD
      IF(NVAL.GT.1)GO TO 50
      GO TO 100
C
C-------------------------------------------------------------------------
C-THE ADDRESS CONTAINS A VALUE ON A MULTIVALUE LIST
 35   NEWLOC=REGASP
      IF(REGASP.EQ.LSTSPC(REGASP)) GO TO 98
C
C   UPDATE AVAILABLE SPACE AND REGASP
      CALL LVUPDT
C
C   MOVE THE VALUE ON A MULTIVALUE LIST TO THE FIRST CELL OF AVAILABLE S
      NODSPC(NEWLOC)=NODSPC(IADD)
      LSTSPC(NEWLOC)=LSTSPC(IADD)
      LNKSPC(NEWLOC)=LNKSPC(IADD)
      FLGSPC(NEWLOC)=FLGSPC(IADD)
```

```
      FLGSPC(IADD)=0
C
C     RESET POINTERS
C
      L1=LSTSPC(NEWLOC)
      IF((FL0MSK.AND.FLGSPC(L1)).EQ.0) GO TO 200
      LNKSPC(LSTSPC(L1))=NEWLOC
      GO TO 201
 200  LNKSPC(L1)=NEWLOC
 201  KZVAL=LSTSPC(LNKSPC(NEWLOC))
      IF((FLGSPC(KZVAL).AND.FL0MSK).NE.0)  GO TO 38
      LSTSPC(LNKSPC(NEWLOC))=NEWLOC
      GO TO 39
  38  LSTSPC(KZVAL)=NEWLOC
  39  NODSPC(IADD)=IFUNC
C   INSERT THIS FUNCTION AS THE HEAD OF A CONFLICT LIST
      LNKSPC(IADD)=IADD
      LSTSPC(IADD)=LSTTMP
      FLGSPC(IADD)=FLGSPC(IADD).OR.FLGTMP.OR.FL4MSK.OR.FL5MSK
      GO TO 100
C
C-------------------------------------------------------------------------
C-THE FUNCTION TO BE INSERTED IS ON THE CONFLICT LIST
  43  HEAD=THIS
C
C     IS THIS A SINGLE VALUE LIST OR MULTIVALUE LIST?
      IF(LSTHED.LT.0) GO TO 51
C
C     OLDLOC IS THE LOCATION OF THE LAST VALUE ON THE MULTIVALUE LIST
C
      OLDLOC=LNKSPC(LSTHED)
C
C-------------------------------------------------------------------------
C-INSERT ADDITIONAL FUNCTION VALUES
 50   LSTASP=NODSPC(REGASP)
      IN=0
      GO TO 56
C
C-------------------------------------------------------------------------
C-FORM MULTIVALUE LIST TO ADD VALUE(S) TO SINGLE-VALUED FUNCTION
  51  IN=0
      IF(REGASP.EQ.LSTSPC(REGASP))GO TO 98
      LSTASP=NODSPC(REGASP)
      NEWLOC=REGASP
      REGASP=LSTSPC(REGASP)
      NODSPC(NEWLOC)=LSTSPC(THIS)
      TEMP=(FLGSPC(THIS).AND.FLG67)
      FLGSPC(NEWLOC)=(TEMP.OR.FLGSPC(NEWLOC))
      FLGSPC(THIS)=(FLGSPC(THIS).AND.NFLG67)
      FLGSPC(THIS)=(FL2MSK.OR.FLGSPC(THIS))
      FLGSPC(THIS)=(FL0MSK.OR.FLGSPC(THIS))
      OLDLOC=THIS
C
C-------------------------------------------------------------------------
C   INSERT ANOTHER VALUE ON MULTIVALUE LIST
 52   FLGSPC(NEWLOC)=(FL2MSK.OR.FLGSPC(NEWLOC))
      FLGSPC(NEWLOC)=(FL1MSK.OR.FLGSPC(NEWLOC))
```

```
      LSTSPC(OLDLOC)=NEWLOC
      LNKSPC(NEWLOC)=OLDLOC
      OLDLOC=NEWLOC
 56   NEWLOC=REGASP
      IF(IN.GT.0)GO TO 57
C
C   NO VALUES HAVE BEEN INSERTED YET
      IN=1
      GO TO 58
C
C   SOME VALUES HAVE BEEN INSERTED
 57   IF(IN.EQ.NVAL)GO TO 67
      IN=IN+1
C
   58 IF(REGASP.EQ.LSTSPC(REGASP)) GO TO 909
      REGASP=LSTSPC(REGASP)
      NODSPC(NEWLOC)=IVALS(IN)
      FLGSPC(NEWLOC)=(ITYP1(IN).OR.FLGSPC(NEWLOC))
      ITYP1(IN)=0
      GO TO 52
C
C   END MULTIVALUE LIST AND UPDATE AVAILABLE SPACE
 67   LSTSPC(OLDLOC)=HEAD
      NODSPC(REGASP)=LSTASP
      LSTSPC(LSTASP)=REGASP
      LNKSPC(LSTSPC(HEAD))=OLDLOC
      GO TO 100
C
C-------------------------------------------------------------------------
C-THE FUNCTION TO BE INSERTED IS NOT ON THE CONFLICT LIST
 60   ASPREG=REGASP
      LSTASP=NODSPC(REGASP)
      IF(REGASP.EQ.LSTSPC(REGASP)) GO TO 98
C
C   UPDATE AVAILABLE SPACE AND REGASP
      CALL LVUPDT
C
C   INSERT FUNCTION IN FIRST CELL OF AVAILABLE SPACE
      NODSPC(ASPREG)=IFUNC
      IF(NVAL.EQ.1)GO TO 611
      LSTSPC(ASPREG)=REGASP
      FLGSPC(ASPREG)=(FL2MSK.OR.FLGSPC(ASPREG))
      FLGSPC(ASPREG)=(FL0MSK.OR.FLGSPC(ASPREG))
      GO TO 612
 611  LSTSPC(ASPREG)=IVALS(1)
  612 FLGSPC(ASPREG)=FLGSPC(ASPREG).OR.ITYP1(1).OR.FL1MSK.OR.FL4MSK
      LNKSPC(ASPREG)=IADD
      LNKSPC(LAST)=ASPREG
      IF(NVAL.EQ.1) GO TO 100
C
C   INSERT ADDITIONAL VALUES
      LSTASP=NODSPC(REGASP)
      OLDLOC=ASPREG
      HEAD=ASPREG
      IN=0
      GO TO 56
C
```

```
C      DESTRUCTIVE INSERTION
C
       ENTRY LVDSIN
C
C     A CALL TO LVFIND MUST PRECEDE A CALL TO EITHER ENTRY POINT.
C     GIVEN N VALUES OF TYPE K ON A LIST WHERE N.GE.0 , INDEXED
C     INSERTIONS SHALL SUCCEED FOR IPOS.GE.1 .AND. IPOS.LE.N+1
C
C     DEFEAT SAVED INDEX UNTIL NEXT RETRIEVAL.
       FLGSPC(THIS)=FLGSPC(THIS).OR.FL4MSK
       JPOS=IABS(IPOS)
       KPOS=IPOS
       INDEX=0
C     DOES THE IPOS'TH VALUE OF THE PROPER TYPE EXIST?
       IF(ITESTR.LT.0) GO TO 90
C     REPLACE VALUE AT LOCATION 'LOC'.  SVL OR MVL?
       IF(LSTHED.GT.0) GO TO 356
C     SVL
       SVLRPL=1
       LSTSPC(LOC)=IVALS(1)
       GO TO 365
C     MVL
 356   NODSPC(LOC)=IVALS(1)
C     REPLACE TYPE.
 365   FLGSPC(LOC)=((FLGSPC(LOC).AND.NFLG67).OR.ITYP1(1))
       GO TO 100
C
C     IPOS'TH VALUE WAS NOT FOUND, INDEXED INSERTION CAN STILL SUCCEED
C     IF (IPOS-1) VALUE IS FOUND.  THIS THEN BECOMES A NORMAL INSERTION
C     IF JPOS=1 OR THE VALUE WILL BE THE LAST IN THE LIST.  OTHERWISE,
C     THIS BECOMES A NONDESTRUCTIVE INSERTION TO THE FIRST POSITION IN
C     THE LIST
C
  90   IF(JPOS.EQ.1) GO TO 125
       IF(KPOS) 91,97,92
  91   KPOS=KPOS+1
       GO TO 93
  92   KPOS=KPOS-1
  93   CALL LVFIND
       IPOS=KPOS
       CALL LVFNV(INDEX)
C     FAILURE IF NO VALUE IS FOUND.
       IF(ITESTR.LT.0) GO TO 97
C     NORMAL INSERTION IF REQUEST WAS IPOS'TH FROM THE TOP.
       IF(KPOS.GT.0) GO TO 125
C     NONDESTRUCTIVE INSERTION AT THE BEGINNING OF THE LIST.
       NEWLOC=REGASP
       CALL LVUPDT
C     SVL OR MVL?
       IF(LSTHED.GT.0) GO TO 377
       GO TO 344
C
C      NONDESTRUCTIVE INSERTION
C
       ENTRY LVNDIN
C
C     IF IPOS=-1, PLACE AT THE END OF THE LIST (NORMAL INSERTION).
```

```
      IF(IPOS.EQ.-1) GO TO 125
C
C     DEFEAT SAVED INDEX UNTIL NEXT RETRIEVAL.
      FLGSPC(THIS)=FLGSPC(THIS).OR.FL4MSK
      JPOS=IABS(IPOS)
      KPOS=IPOS
      INDEX=0
      NEWLOC=REGASP
C     DOES THE IPOS'TH VALUE OF THE PROPER TYPE EXIST?
      IF(ITESTR.LT.0) GO TO 90
      CALL LVUPDT
C     SVL OR MVL?
      IF(LSTHED.LT.0) GO TO 344
C     MVL
      IF(KPOS.LT.0) GO TO 347
C
C     PLACE VALUE AT THE IPOS'TH POSITION (WRT ITYP) FROM THE TOP OF LIST
 377  ISTLOC=LNKSPC(LOC)
      NODSPC(NEWLOC)=IVALS(1)
      LSTSPC(NEWLOC)=LOC
      LNKSPC(NEWLOC)=ISTLOC
      FLGSPC(NEWLOC)=FL1MSK.OR.FL2MSK.OR.ITYP1(1)
      IF(LOC.NE.LSTHED) GO TO 321
      LSTSPC(LSTSPC(ISTLOC))=NEWLOC
      GO TO 322
 321  LSTSPC(ISTLOC)=NEWLOC
 322  LNKSPC(LOC)=NEWLOC
      GO TO 100
C
C     PLACE VALUE AT THE IPOS'TH POSITION (WRT ITYP) FROM THE BOTTOM OF
C     THE LIST
 347  NODSPC(NEWLOC)=IVALS(1)
      LSTSPC(NEWLOC)=LSTSPC(LOC)
      LNKSPC(NEWLOC)=LOC
      FLGSPC(NEWLOC)=FL1MSK.OR.FL2MSK.OR.ITYP1(1)
      IF((FLGSPC(LSTSPC( LOC)).AND.FL0MSK).EQ.0) GO TO 323
      KZVAL=LSTSPC(LOC)
      LNKSPC(LSTSPC(KZVAL))=NEWLOC
      GO TO 324
 323  LNKSPC(LSTSPC( LOC))=NEWLOC
 324  LSTSPC( LOC)=NEWLOC
      GO TO 100
C
C     CREATE MVL WITH NEW VALUE AT THE TOP OF THE LIST.
 344  IF(REGASP.EQ.LSTSPC(REGASP)) GO TO 99
      NWLOC2=REGASP
      CALL LVUPDT
      NODSPC(NEWLOC)=IVALS(1)
      LSTSPC(NEWLOC)=NWLOC2
      LNKSPC(NEWLOC)=NWLOC2
      FLGSPC(NEWLOC)=FL1MSK.OR.FL2MSK.OR.ITYP1(1)
      NODSPC(NWLOC2)=LSTSPC(THIS)
      LSTSPC(NWLOC2)=THIS
      LNKSPC(NWLOC2)=NEWLOC
      KLGTEP=FLGSPC(THIS).AND.FLG67
      FLGSPC(NWLOC2)=(FL1MSK.OR.FL2MSK).OR.KLGTEP
      LSTSPC(THIS)=NEWLOC
```

```
      FLGSPC(THIS)=(FLGSPC(THIS).OR.FL0MSK).OR.FL2MSK
      GO TO 100
   98 ITESTR=-4
      PRINT 20001
20001 FORMAT(  * ERROR...THERE IS NO ADDITIONAL SPACE FOR THE GRAPH, THE
     * PROGRAM IS TERMINATED*)
      STOP
   99 ITESTR=-3
      PRINT 2, NVAL
  2   FORMAT(6H ONLY ,I4,28H VALUE(S) HAVE BEEN INSERTED)
  22  FORMAT(1X,I5,1H(,I5,35H) USED LAST CELL OF AVAILABLE SPACE)
      GO TO 97
  909 PRINT 22,IFUNC,IARG
C     THIS INSERTION HAS FILLED GIRS MEMORY - CALL A USER SUPPLIED
C     PROGRAM - LVEXIT.
      ITESTR=-2
      GO TO 97
  100 IF(REGASP.EQ.LSTSPC(REGASP)) GO TO 909
C     FLAG 4 IS SET BECAUSE THIS INSERTION MIGHT BE A RECREATION OF AN
C     OLD LIST
      FLGSPC(THIS)=FLGSPC(THIS).OR.FL4MSK
      IVAL=IVALS(1)
C     "FAILURE" RETURN IF FUNCTION DID NOT PREVIOUSLY EXIST
      IF(((FLGSPC(THIS).AND.FL0MSK).NE.0).OR.SVLRPL.EQ.1) ITESTR=1
   97 IPOS=1
      ITYP=3
      NVAL=1
      SVLRPL=0
      ITYP1(1)=0
      RETURN
      END



      SUBROUTINE LVUPDT
      INTEGER REGASP,FLGSPC
      COMMON/LVVTR1/MEMSZE,REGASP,NODSPC(    1)/LVVTR2/LSTSPC(    1)/
     *LVVTR3/LNKSPC(    1)/LVVTR4/FLGSPC(    1)
C
C     THIS ROUTINE UPDATES AVAILABLE SPACE AND THE REGISTER OF AVAILABLE
C     SPACE - REGASP
C
      LSTSPC(NODSPC(REGASP))=LSTSPC(REGASP)
      NODSPC(LSTSPC(REGASP))=NODSPC(REGASP)
      REGASP=LSTSPC(REGASP)
      RETURN
      END
```

```
      SUBROUTINE LVDLET
      INTEGER FLGSPC,REGASP,FL0MSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,
     + FLG67,SEQSPC,THIS
      COMMON/LVVTR1/MEMSZE,REGASP,NODSPC(   1)/LVVTR2/LSTSPC(   1)/
     *LVVTR3/LNKSPC(   1)/LVVTR4/FLGSPC(   1)
      COMMON /LVADDR/ IADD,THIS,LSTHED,LOC,LAST
      COMMON /LVARGS/IFUNC,IARG,IPOS,ITYP,IVAL,NVAL,NSKIP,ITESTR,INCLUD,
     + IVALS(10),ITYP1(10)
      COMMON/LVFLAG/FL0MSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,FLG67
      COMMON /LVVSEQ/ISEQSZ,ISQPOS,LASTSQ,SEQSPC(1)
      DATA NFLG02/137B/
C
C     DELETE ENTIRE LIST.  CALL FIND TO DETERMINE SVL OR MVL, LOCATION
C     OF FUNCTION AND FIRST VALUE.  FAILURE RETURN IF NO LIST.
C
      CALL LVFIND
      IF(ITESTR.LT.0) RETURN
      IF(LSTHED.LT.0) GO TO 2
C     DELETE ENTIRE MULTIVALUE LIST
      ISADD=LSTHED
      LOC=THIS
    5 NXTADD=LSTSPC(ISADD)
      NODSPC(ISADD)=NODSPC(REGASP)
      LSTSPC(ISADD)=REGASP
      LNKSPC(ISADD)=0
      FLGSPC(ISADD)=0
      LSTSPC(NODSPC(REGASP))=ISADD
      NODSPC(REGASP)=ISADD
      IF((FLGSPC(NXTADD).AND.FL0MSK).NE.0) GO TO 2
      ISADD=NXTADD
      GO TO 5
C
C     DELETE SINGLE VALUED FUNCTION
C     IS THE FUNCTION HEAD OF A CONFLICT LIST
    2 IF(THIS.NE.IADD) GO TO 68
      NXFUNC=LNKSPC(IADD)
C     IF THIS FUNCTION IS THE ONLY ONE ON THE CONFLICT LIST, GO TO 10.
C     OTHERWISE, PLACE NEXT FUNCTION ON CONFLICT LIST IN 'HEAD OF
C     CONFLICT LIST' LOCATION (IADD)
      IF(NXFUNC.EQ.IADD) GO TO 10
      NODSPC(IADD)=NODSPC(NXFUNC)
      LSTSPC(IADD)=LSTSPC(NXFUNC)
      LNKSPC(IADD)=LNKSPC(NXFUNC)
      FLGSPC(IADD)=FLGSPC(NXFUNC)
      FLGSPC(IADD)=FLGSPC(IADD).OR.FL5MSK
      IF((FLGSPC(IADD).AND.FL0MSK).EQ.0) GO TO 9
C     IF THE MOVED FUNCTION IS A MVL, THE POINTER FROM THE LAST VALUE OF
C     THE LIST TO THE HEAD MUST BE UPDATED.
      KVAL=LSTSPC(IADD)
    8 KVAL=LSTSPC(KVAL)
      IF((FLGSPC(LSTSPC(KVAL)).AND.FL0MSK).EQ.0)  GO TO 8
      LSTSPC(KVAL)=IADD
    9 LOC=NXFUNC
C     RETURN LOCATION TO AVAILABLE SPACE
   10 NODSPC( LOC)=NODSPC(REGASP)
      LSTSPC( LOC)=REGASP
      LNKSPC( LOC)=0
```

```
      FLGSPC( LOC)=0
      NODSPC(LSTSPC( LOC))=LOC
      LSTSPC(NODSPC( LOC))=LOC
      RETURN
C
C     FUNCTION TO BE DELETED IS NOT THE HEAD OF A CONFLICT LIST.
C     THE FUNCTION PRECEDING THIS (FUNCTION BEING DELETED) MUST POINT TO
C     THE FUNCTION FOLLOWING THIS
   68 LNKSPC(LAST)=LNKSPC(THIS)
      GO TO 10
C
      ENTRY LVDLTI
C
C     THIS ENTRY POINT WILL HANDLE INDEXED DELETION.
C
C     FUNCTION MUST BE A MVL OR, IF SVL, ABS(IPOS)=1 WITH PROPER TYPE.
C     OUTPUT IS EXPECTED FROM LVFIND.
C     DOES THE FUNCTION EXIST ?
      IF(ITESTR.LT.0) RETURN
C     SVL OR MVL ?
      IF(LSTHED.LT.0) GO TO 2
C     DELETE VALUE AT LOC.  DEFEAT SAVED INDEX FOR THIS LIST UNTIL AFTER
C     NEXT RETRIEVAL.
      FLGSPC(THIS)=FLGSPC(THIS).OR.FL4MSK
C
C     INDEXED DELETE CAN BE REDUCED TO FOUR CASES.  DELETE VALUE IN
C     FIRST, MIDDLE, OR LAST POSITION ON LIST, OR REDUCE TO SVL.
C
      NEXT=LSTSPC(LOC)
      NPRIOR=LNKSPC(LOC)
C
C     IS LOC THE  LAST POSITION IN THE LIST ?
      IF(NEXT.EQ.THIS) GO TO 80
C
C     IS LOC THE FIRST POSITION IN THE LIST ?
      IF(LSTSPC(NPRIOR).EQ.THIS) GO TO 70
C
C     VALUE IS IN A MIDDLE POSITION IN THE LIST.  RECONNECT VALUES
C     PRECEEING AND FOLLOWING LOC.
C
      LSTSPC(NPRIOR)=NEXT
      LNKSPC(NEXT)=NPRIOR
      GO TO 10
C
C     DELETE VALUE IN LAST POSITION IN LIST
   80 LSTSPC(NPRIOR)=NEXT
      NEXT1=LSTSPC(NEXT)
      LNKSPC(NEXT1)=NPRIOR
      GO TO 60
C
C     DELETE VALUE IN FIRST POSITION IN LIST
   70 LNKSPC(NEXT)=NPRIOR
      LSTSPC(THIS)=NEXT
C
C     CONVERT TO A SINGLE VALUE LIST ?
C
   60 IF(LNKSPC(NPRIOR).NE.NPRIOR) GO TO 10
```

```
C     IF DELETING LAST VALUE, RESET NEXT TO FIRST VALUE
      IF(NEXT.EQ.THIS) NEXT=NPRIOR
      LSTSPC(THIS)=NODSPC( NEXT)
      FLGSPC(THIS)=(FLGSPC(THIS).OR.FLGSPC( NEXT)).AND.NFLG02
      FLGSPC( NEXT)=0
      LNKSPC( NEXT)=0
      NODSPC( NEXT)=NODSPC(REGASP)
      LSTSPC( NEXT)=REGASP
      NODSPC(LSTSPC( NEXT))=NEXT
      LSTSPC(NODSPC( NEXT))=NEXT
      GO TO 10
      END
```

```
      SUBROUTINE LVDUMP(KK,JJ,L)
      INTEGER FLGSPC,FLAGSP,BINFIL,SEQSPC,REGASP
      COMMON/LVVTR1/MEMSZE,REGASP,NODSPC(    1)/LVVTR2/LSTSPC(    1)/
     *LVVTR3/LNKSPC(    1)/LVVTR4/FLGSPC(    1)
      COMMON/LVRAND/ KPRIME,KSEED,NROW,KDNODE,KDROW,KTEMP
      COMMON /LVVSEQ/ISEQSZ,ISQPOS,LASTSQ,SEQSPC(1)
      COMMON/LVVTR5/BINFIL,KOMPAN,NODESP(1)/LVVTR6/LISTSP(1)
     + /LVVTR7/LINKSP(1)/LVVTR8/FLAGSP(1)
      COMMON /LVTABL/ MAPSZE,MAP(1)
      IF(JJ.EQ.0) GO TO 50
      K=KK
      J=JJ
      IF(KK.LT.1) K=1
      IF(JJ.GT.MEMSZE) J=MEMSZE
      WRITE(L,10)
10    FORMAT(1H1,*       GIRS MEMORY DUMP (IN OCTAL)*,///)
      WRITE(L,20) REGASP,MEMSZE,KPRIME,KSEED,NROW,KDNODE,KTEMP,KDROW,
     + ISEQSZ,MAPSZE
20    FORMAT(1X,* REGASP=*,I6,/* MEMSZE=*,I6,6X,*PRIME=*,I3,6X,*SEED=*
     +,I3,6X,*NROW=*,I6,6X,6X,*KDNODE=*,I6,6X,*TEMP=*,I6,6X,
     +*KDROW=*,I6,6X,/,1X,*SEQSIZE=*,I6,6X,*MAPSIZE=*,I6,///)
      WRITE(L,30)
30    FORMAT(1X,*                  NODSPC                  LSTSPC
     *     LNKSPC          FLGSPC  OCTAL COUNTER*,///)
      WRITE(L,15)(I,NODSPC(I),LSTSPC(I),LNKSPC(I),FLGSPC(I),I,I=K,J)
15    FORMAT(1X,I6,2X,O20,2X,O20,2X,O20,2X,O6,2X,O8)
      RETURN
50    WRITE(L) REGASP,MEMSZE,KPRIME,KSEED,NROW,KDNODE,KTEMP,KDROW,
     + ISEQSZ,MAPSZE
      WRITE(L)(NODSPC(M),M=1,MEMSZE)
      WRITE(L)(LSTSPC(M),M=1,MEMSZE)
      WRITE(L)(LNKSPC(M),M=1,MEMSZE)
      WRITE(L)(FLGSPC(M),M=1,MEMSZE)
      WRITE(L)(SEQSPC(I),I=1,ISEQSZ)
      RETURN
      END
```

```
      SUBROUTINE LVCMPN
      IMPLICIT INTEGER(A-Z)
      COMMON/LVVTR1/MEMSZE,REGASP,NODSPC(1)
     1 /LVVTR2/LSTSPC(1)
     1 /LVVTR3/LNKSPC(1)
     1 /LVVTR4/FLGSPC(1)
      COMMON /LVADDR/ IADD,THIS,LSTMEC,LOC,LAST
      COMMON/LVVTR5/ BINFIL,KOMPAN,NODESP(1)/LVVTR6/LISTSP(1)
     + /LVVTR7/LINKSP(1)/LVVTR8/FLAGSP(1)
      COMMON /LVTABL/ MAPSZE,IEXTRA,MAP(1)
      COMMON /LVARGS/IFUNC,IARG,IPOS,ITYP,IVAL,NVAL,NSKIP,ITESTR,INCLUD,
     + INDXCN,IVALS(10),ITYP1(10)
      COMMON/LVRAND/ KPRIME,KSEED,NROW,KONODE,KDROW,KTEMP
      COMMON/LVFLAG/FLOMSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,FLG67
      COMMON /LVVSEQ/ISEQSZ,ISQPOS,LASTSQ,SEQSPC(1)
      DIMENSION IPRIME(24),LRAND(6)
      DATA IPRIME/3,5,7,11,13,17,19,23,29,31,37,41,43,
     + 47,53,59,61,67,71,73,79,83,89,97/
C     GO TO 10 IF SECOND CALL FROM LVSETP
      IF(KOMPAN.EQ.3) GO TO 10
C
C     READ IN OLD GRAPH, NEW GRAPH TO BE PLACED IN NODSPC, LSTSPC, LNKSPC, FLGSPC
      READ(BINFIL) LEGASP,OLDMEM,(LRAND(I),I=1,6),ISEQSZ,MPSZE
      READ(BINFIL)(NODESP(I),I=1,OLDMEM)
      READ(BINFIL)(LISTSP(I),I=1,OLDMEM)
      READ(BINFIL)(LINKSP(I),I=1,OLDMEM)
      READ(BINFIL)(FLAGSP(I),I=1,OLDMEM)
      READ(BINFIL)(SEQSPC(I),I=1,ISEQSZ)
C
C  DETERMINE MINIMUM BUFFER SIZE
C  2 CRITERIA MUST BE MET.
C   A) THE BUFFER MUST BE ABLE TO HOLD ALL OF THE TRIPLETS PLUS ONE SPACE
C   B) THE BUFFER SIZE (MEMSZE) MUST BE LARGE ENOUGH TO DEFINE ALL OF THE
C      NODES AND LINKS
C   A)
      LSUMA=1
      DO 20 I=1,OLDMEM
      IF((FLAGSP(I).AND.FL1MSK).NE.0) LSUMA=LSUMA+1
20    CONTINUE
C
C   B) SET UP REFERENCE TABLE TO DETERMINE MINIMUM NUMBER OF NODES AND
C      LINKS TO BE DEFINED
      DO 15 K=1,MAPSZE
15    MAP(K)=0
      DO 95 J=1,OLDMEM
      I=J
      IF((FLAGSP(I).AND.FL5MSK).EQ.0) GO TO 95
      ADDRES =I
100   LINK=NODESP(I)
      NODE=ADDRES-LINK
      IF(NODE.LE.0)NODE=NODE+OLDMEM
      MAP(LINK)=1
      MAP(NODE)=1
C  MUST SINK NODES BE DEFINED*
C    SVL OR MVL*
      IF((FLAGSP(I).AND.FLOMSK).NE.0) GO TO 90
C    ADD TO TABLE IF RANDOM NUMBER
```

```
      IF((FLAGSP(I).AND.FLG67).EQ.0) MAP(LISTSP(I))=1
      GO TO 80
C
C     MVL
90    I1=I
91    CONTINUE
      I1=LISTSP(I1)
      IF((FLAGSP(I1).AND.FLOWSK).NE.0) GO TO 80
      IF((FLAGSP(I1).AND.FLG67).EQ.0) MAP(NODESP(I1))=1
      GO TO 91
80    I=LINKSP(I)
      IF(I.NE.J) GO TO 100
95    CONTINUE
C
COUNT UP TOTAL NUMBER OF NODES AND LINKS PLUS ONE FREE SPACE
      LSUMB=1
      DO 101 I=1,MAPSZE
      IF(MAP(I).EQ.1) LSUMB=LSUMB+1
101   CONTINUE
C  DETERMINE MINIMUM BUFFER SIZE- LSUMA OR LSUMB
C
      MAX=LSUMA
      IF(LSUMB.GT.MAX) MAX=LSUMB
C
C  IS REQUESTED BUFFER SIZE LARGE ENOUGH*
      IF(MEMSZE.GE.MAX) GO TO 97
C  IF REQUESTED MEMSZE IS TOO SMALL, PRINT WARNING AND CORRECT
      IF(KOMPAN.EQ.2) GO TO 98
      TYPE 99,MEMSZE,MAX
99    FORMAT(, **** WARNING ****, REQUESTED BUFFER SIZE,,,I5,/
     1 ,  IS TOO SMALL AND SHOULD BE REPLACED BY AT LEAST,,I5)
      STOP
98    MEMSZE=MAX+IEXTRA
C
CALCULATE NEW PRIME NUMBER AND ASSOCIATED GRL VARIABLES
      VVSZE=MEMSZE
      PRIME=SQRT(VVSZE)*0.5
      KPRIME=PRIME
      DO 3 KK=1,24
      II=KK
      IF(IPRIME(II).GE.KPRIME) GO TO 14
3     CONTINUE
14    KPRIME=IPRIME(II)
C     SET KOMPAN FOR SECOND CALL TO THIS ROUTINE
97    KOMPAN=3
      RETURN
C
C    SET UP CLD TO NEW RANDOM NUMBER CORRESPONDENCE MAP
10    DO 35 I=1,MAPSZE
      IF(MAP(I).EQ.0) GO TO 35
      CALL LVGRN(INDEX)
      MAP(I)=INDEX
35    CONTINUE
C
CONVERT GRAPH
C     SEARCH FOR HEADS OF CONFLICT LISTS AND PLACE ALL LISTS RELATED TO
C     THAT ADDRESS INTO THE NEW GRAPH
```

```
      DO 45 J=1,OLDMEM
      I=J
      IF((FLAGSP(I).AND.FLSMSK).EQ.0) GO TO 45
      ADDRES =I
60    LINK=NODESP(I)
      IFUNC=MAP(LINK)
      NODE=ADDRES-LINK
      IF(NODE.LE.0)NODE=NODE+OLDMEM
      IARG=MAP(NODE)
C     SVL OR MVL*
      IF((FLAGSP(I).AND.FLOMSK).NE.0) GO TO 50
C     OBTAIN NEW ADDRESS FOR NEW GRAPH
      CALL LVFIND
      IVALS(1)=LISTSP(I)
      ITYP1(1)=FLAGSP(I).AND.FLG67
      IF(ITYP1(1).EQ.0) IVALS(1)=MAP(IVALS(1))
      CALL LVNSRT
      GO TO 40
C
C     PLACE MULTIVALUE LIST INTO NEW GRAPH
50    I1=I
51    CONTINUE
      I1=LISTSP(I1)
      IF((FLAGSP(I1).AND.FLOMSK).NE.0) GO TO 40
      CALL LVFIND
      IVALS(1)=NODESP(I1)
      ITYP1(1)=FLAGSP(I1).AND.FLG67
      IF(ITYP1(1).EQ.0) IVALS(1)=MAP(IVALS(1))
      CALL LVNSRT
      GO TO 51
40    I=LINKSP(I)
      IF(I.NE.J) GO TO 60
45    CONTINUE
      TYPE 70,NEWSZE
70    FORMAT(' NEW GRAPH HAS BEEN CREATED - NEW SIZE IS ',I6)
      RETURN
      END
```

```
      SUBROUTINE LVUNPK(L)
      INTEGER FLGSPC,REGASP,FL0MSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,
     + FLG67,SEQSPC,THIS
      COMMON/LVVTR1/MEMSZE,REGASP,NODSPC(    1)/LVVTR2/LSTSPC(    1)/
     *LVVTR3/LNKSPC(    1)/LVVTR4/FLGSPC(    1)
      COMMON/LVVTR5/BINFIL,KOMPAN,NODESP(1)/LVVTR6/LISTSP(1)
     + /LVVTR7/LINKSP(1)/LVVTR8/FLAGSP(1)
      COMMON /LVTABL/ MAPSZE,MAP(1)
      COMMON /LVVSEQ/ISEQSZ,ISQPOS,LASTSQ,SEQSPC(1)
      COMMON/LVRAND/ KPRIME,KSEED,NROW,KONODE,KDROW,KTEMP
      COMMON/LVFLAG/FL0MSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,FLG67
      COMMON /LVMASK/ MASK1,MASK2,MASK3,MASK4,NMASK4
      COMMON /LVSHFT/ ISHFT1,ISHFT2,ISHFT3
C
C     THIS ROUTINE UNPACKS A GIRS BUFFER WHICH WAS CREATED WITH
C     THE PACKED VERSION
C
      READ(L) REGASP,MEMSZE,KPRIME,KSEED,NROW,KONODE,KTEMP,KDROW,
     + ISEQSZ,MAPSZE
      READ(L)(FLGSPC(I),I=1,MEMSZE)
      READ(L)(SEQSPC(I),I=1,ISEQSZ)
      DO 30 I=1,MEMSZE
      KTEMP1=FLGSPC(I).AND.MASK1
      KTEMP2=FLGSPC(I).AND.MASK2
      KTEMP3=FLGSPC(I).AND.MASK3
      FLGSPC(I)=FLGSPC(I).AND.MASK4
      NODSPC(I)=LVRTSH(KTEMP1,ISHFT1)
      LSTSPC(I)=LVRTSH(KTEMP2,ISHFT2)
      LNKSPC(I)=LVRTSH(KTEMP3,ISHFT3)
   30 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE LVSTAK
      COMMON /LVARGS/IFUNC,IARG,IPOS,ITYP,IVAL,NVAL,NSKIP,ITESTR,INCLUD,
     + IVALS(10),ITYP1(10)
      COMMON /LVSTAC/ ISTACK,MAXLEV,LSTACK(1)
C
C     THIS ROUTINE STORES ALL PERTINENT ARGUMENTS WHEN GIRL STATEMENTS
C     ARE PARENTHESIZED.  UP TO FIVE LEVELS OF PARENTHESIZATION ARE
C     ALLOWED ON DEFAULT.  STOP IF THE MAXIMUM NUMBER OF LEVELS HAVE BEEN
C     EXCEEDED.
C
      IF(ISTACK.GE.MAXLEV) GO TO 99
      IF(ISTACK.LT.0) GO TO 98
      LSTACK(ISTACK+1) = IFUNC
      LSTACK(ISTACK+2) = IARG
      LSTACK(ISTACK+3) = IPOS
      LSTACK(ISTACK+4) = ITYP
      LSTACK(ISTACK+5) = IVAL
      LSTACK(ISTACK+6) = ITESTR
      LSTACK(ISTACK+7) = INCLUD
      LSTACK(ISTACK+8) = IVALS(1)
      LSTACK(ISTACK+9) = ITYP1(1)
      ISTACK=ISTACK+9
      RETURN
C
C     MAXIMUM NUMBER OF PARENTHISIZED LEVELS EXCEEDED, STOP
C
   99 PRINT 1
    1 FORMAT(*  ERROR... MAXIMUM NUMBER OF PARENTHESIZED LEVELS HAS BEEN
     + EXCEEDED, STOP*)
      STOP
C
   98 PRINT 2
    2 FORMAT(* ERROR... NUMBER OF RIGHT PARENTHESES EXCEEDS NUMBER OF LE
     +FT PARENTHESES, STOP*)
      STOP
      END
```

```
      SUBROUTINE LVPOP
      COMMON /LVARGS/IFUNC,IARG,IPOS,ITYP,IVAL,NVAL,NSKIP,ITESTR,INCLUD,
     + IVALS(10),ITYP1(10)
      COMMON /LVSTAC/ ISTACK,MAXLEV,LSTACK(1)
C
C     THIS ROUTINE OUTPUTS THE TOP NINE VALUES ON THE ARGUMENT STACK UPON
C     HITTING A COMMA IN A GIRL STATEMENT.
      IF(ISTACK.LT.9) GO TO 99
      IFUNC    = LSTACK(ISTACK-8)
      IARG     = LSTACK(ISTACK-7)
      IPOS     = LSTACK(ISTACK-6)
      ITYP     = LSTACK(ISTACK-5)
      IVAL     = LSTACK(ISTACK-4)
      ITESTR   = LSTACK(ISTACK-3)
      INCLUD   = LSTACK(ISTACK-2)
      IVALS(1)= LSTACK(ISTACK-1)
      ITYP1(1)= LSTACK(ISTACK)
      RETURN
C
   99 PRINT 1
    1 FORMAT(*  ERROR... ATTEMPT TO POP EMPTY STACK, STOP*)
      STOP
      END
```

```
      SUBROUTINE LVSETP
      INTEGER WRKSPC,WORKSP,BINFIL,SEQSPC,REGASP
      COMMON/LVVTR1/MEMSZE,REGASP,WRKSPC(1)
      COMMON/LVRAND/ KPRIME,KSEED,NROW,KDNODE,KDROW,KTEMP
      COMMON/LVVTR5/BINFIL,KOMPAN,WORKSP(1)
      COMMON/LVMASK/MASK1,MASK2,MASK3,MASK4,NMASK1,NMASK2,NMASK3,NMASK4
      COMMON /LVSHFT/ ISHFT1,ISHFT2,ISHFT3
      IF(KOMPAN.NE.0) CALL LVCMPN
      KSEED=KPRIME/2
      NROW=KSEED
      KTEMP=KSEED-KPRIME
      KDNODE=KPRIME
      CALL LVGRN(REGASP)
      IOLD=REGASP
      DO 10 I=2,MEMSZE
      CALL LVGRN(NEW)
      IOLDTM=LVLFSH(IOLD,ISHFT1)
      NEWTMP=LVLFSH( NEW,ISHFT2)
      WRKSPC( NEW)=(WRKSPC( NEW).AND.NMASK1).OR.IOLDTM
      WRKSPC(IOLD)=(WRKSPC(IOLD).AND.NMASK2).OR.NEWTMP
10    IOLD=NEW
      IOLDTM=LVLFSH(IOLD,ISHFT1)
      WRKSPC(REGASP)=(WRKSPC(REGASP).AND.NMASK1).OR.IOLDTM
      WRKSPC(IOLD)=(WRKSPC(IOLD).AND.NMASK2).OR.LVLFSH(REGASP,ISHFT2)
      NROW=KSEED
      KTEMP=KSEED-KPRIME
      KDNODE=KPRIME
      IF(KOMPAN.NE.0) CALL LVCMPN
      RETURN
      END
```

```
      SUBROUTINE LVFECH(N)
      INTEGER WRKSPC,REGASP,SEQSPC
      COMMON/LVVTR1/MEMSZE,REGASP,WRKSPC(1)
      COMMON/LVFLAG/FL0MSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,FLG67
      COMMON /LVVSEQ/ISEQSZ,ISQPOS,LASTSQ,SEQSPC(1)
      COMMON/LVRAND/ KPRIME,KSEED,NROW,KDNODE,KDROW,KTEMP
      READ(N) REGASP,MEMSZE,KPRIME,KSEED,NROW,KDNODE,KTEMP,KDROW,
     + ISEQSZ,MAPSZE
      READ(N)(WRKSPC(I),I=1,MEMSZE)
      READ(N)(SEQSPC(I),I=1,ISEQSZ)
      PRINT 10
10    FORMAT(1H ,*  GRAPH HAS BEEN PLACED INTO MEMORY*,//)
      RETURN
      END
```

```
      SUBROUTINE LVFIND
      INTEGER WRKSPC,REGASP,FL0MSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,
     + FLG67,SEQSPC,THIS
      COMMON/LVVTR1/MEMSZE,REGASP,WRKSPC(1)
      COMMON /LVADDR/ IADD,THIS,LSTHED,LOC,LAST
      COMMON /LVARGS/IFUNC,IARG,IPOS,ITYP,IVAL,NVAL,NSKIP,ITESTR,INCLUD,
     + IVALS(10),ITYP1(10)
      COMMON/LVFLAG/FL0MSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,FLG67
      COMMON /LVVSEQ/ISEQSZ,ISQPOS,LASTSQ,SEQSPC(1)
      COMMON/LVMASK/MASK1,MASK2,MASK3,MASK4,NMASK1,NMASK2,NMASK3,NMASK4
      COMMON /LVSHFT/ ISHFT1,ISHFT2,ISHFT3
      DATA FL0MSK/200B/,FL1MSK/100B/,FL2MSK/40B/,FL5MSK/4B/,FLG67/3B/,
     + FL3MSK/20B/,FL4MSK/10B/
      DATA MASK1/777777000000000000B/,MASK2/777777000000B/,
     + MASK3/777770000B/,MASK4/7778/,NMASK1/777777777777778/,
     + NMASK2/777777000000777777778/,NMASK3/777777777777000007778/,
     + NMASK4/777777777777777770008/
      DATA ISHFT1,ISHFT2,ISHFT3/42,24,9/
C
C     IADD   = COMPUTED FUNCTION ADDRESS
C     THIS   = LOCATION OF FUNCTION ON CONFLICT LIST
C     LOC    = LOCATION OF RETRIEVED VALUE
C     LSTHED = -1, SINGLE VALUED LIST
C            = 0, NO LIST IS FOUND
C            = .GT.0, ADDRESS OF FIRST VALUE
C     ITESTR = 1, RETRIEVAL IS SUCCESSFUL (IVAL = RETURNED VALUE)
C            = -1, RETRIEVAL IS FAILURE (IVAL = SOURCE NODE)
      ITESTR=1
      IADD=IFUNC+IARG
      IF(IADD.GT.MEMSZE) IADD=IADD-MEMSZE
      LSTHED=0
      THIS=IADD
      IF((WRKSPC(THIS).AND.FL5MSK).EQ.0) GO TO 99
C     SEARCH CONFLICT LIST FOR KEY (IFUNC OR LINK)
    1 KTEMP=LVRTSH(WRKSPC(THIS),ISHFT1)
      IF(KTEMP.EQ.IFUNC)  GO TO 4
      LAST=THIS
      KTEMP=WRKSPC(THIS).AND.MASK3
      THIS=LVRTSH(KTEMP,ISHFT3)
      IF((WRKSPC(THIS).AND.FL5MSK).NE.0) GO TO 99
      GO TO 1
C
C     THE FUNCTION HAS BEEN FOUND.
C     TEST FOR SINGLE VALUE LIST (SVL)  OR MULTIVALUED LIST (MVL).
    4 IF((WRKSPC(THIS).AND.FL0MSK).NE.0)  GO TO 14
C     SINGLE VALUED LIST.
      LSTHED=-1
      LOC=THIS
      IVAL  =LVRTSH((WRKSPC( LOC).AND.MASK2),ISHFT2)
      GO TO 5
C     MULTIVALUED LIST.  OBTAIN FIRST VALUE.
   14 LSTHED=LVRTSH((WRKSPC(THIS).AND.MASK2),ISHFT2)
      LOC=LSTHED
      IVAL  =LVRTSH (WRKSPC( LOC),ISHFT1)
      GO TO 5
   99 ITESTR=-1
      IVAL=IARG
C     RESET TO DEFAULT VALUES
    5 IPOS=1
      ITYP=3
C     IS IVAL NEGATIVE?
      IVAL=SHIFT(IVAL, ISHFT1)
      IVAL=SHIFT(IVAL,-ISHFT1)
      RETURN
      END
```

```
      SUBROUTINE LVFNV(INDEXS)
      INTEGER WRKSPC,REGASP,FL0MSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,
     + FLG67,SEQSPC,THIS
      COMMON/LVVTR1/MEMSZE,REGASP,WRKSPC(1)
      COMMON /LVADDR/ IADD,THIS,LSTHED,LOC,LAST
      COMMON /LVARGS/IFUNC,IARG,IPOS,ITYP,IVAL,NVAL,NSKIP,ITESTR,INCLUD,
     + IVALS(10),ITYP1(10)
      COMMON/LVFLAG/FL0MSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,FLG67
      COMMON /LVVSEQ/ISEQSZ,ISQPOS,LASTSQ,SEQSPC(1)
      COMMON/LVMASK/MASK1,MASK2,MASK3,MASK4,NMASK1,NMASK2,NMASK3,NMASK4
      COMMON /LVSHFT/ ISHFT1,ISHFT2,ISHFT3
      DATA NFLAG4/7777777777777777678/
C
C     LVFIND MUST BE CALLED IMMEDIATELY PRIOR TO THE CALL TO THIS ROUTINE
C     INPUT IS EXPECTED THRU COMMONS LVARGS AND LVADDR.  THIS ROUTINE
C     SEARCHES THE MULTIVALUE LIST FOR THE IPOS'TH VALUE OF THE REQUESTED
C     TYPE.  IF SVL, TYPE MUST BE EITHER UNSPECIFIED OR CORRECT.
C
C     DOES THE FUNCTION EXIST ?
      IF(ITESTR.LT.0) GO TO 70
      JPOS=IABS(IPOS)
      IF(LSTHED.GT.0) GO TO 60
C     SVL -  DOES FUNCTION QUALIFY ?
      IF(JPOS.NE.1) GO TO 99
      IF(ITYP.EQ.3) GO TO 70
      ISTYP=(WRKSPC( LOC).AND.FLG67)
      IF(ISTYP.EQ.3)ISTYP=2
      IF(ISTYP.NE.ITYP)  GO TO 99
      GO TO 70
C
C     MVL - FIRST VALUE HAS ALREADY BEEN FOUND BY LVFIND, SET INDEX
C     PARAMETERS.
   60 IF(IPOS .EQ. 1 .AND. ITYP .EQ. 3) GO TO 75
C
C     BEGIN SEARCH
      IND=0
C     IF THE SAVED INDEX FACILITY IS NOT TO BE USED, GO TO 50
      IF(NSKIP.EQ.1) GO TO 50
      IF(INDEXS.EQ.0) GO TO 50
C
C     UNPACK THE INPUT PARAMETERS FOR SAVED INDEX
C     INDEX = POSITION FROM TOP OR BOTTOM OF PREVIOUSLY RETRIEVED VALUE
C     INDXAD= LOCATION OF PREVIOUSLY RETRIEVED VALUE
C     KFUNC = FUNCTION CONTAINING PREVIOUSLY RETRIEVED VALUE
C     KARG  = ARGUMENT CONTAINING PREVIOUSLY RETRIEVED VALUE
      KARG  = LVRTSH((INDEXS.AND.MASK1),ISHFT1)
      KFUNC = LVRTSH((INDEXS.AND.MASK2),ISHFT2)
      INDXAD= LVRTSH((INDEXS.AND.MASK3),ISHFT3)
      INDEX =        (INDEXS.AND.MASK4)
C     IS THE INDEX NEGATIVE ?
      IF(INDEX.GE.256) INDEX=INDEX.OR.NMASK4
C
C     SAVED INDEX CAN'T BE USED IF IMMEDIATE PAST HISTORY = INDEXED
C     INSERTION OR DELETION.
      IF((WRKSPC(THIS) .AND. FL4MSK) .NE. 0) GO TO 50
C
C     SAVED INDEX CAN'T BE USED IF SOURCE NODE OR LINK HAVE BEEN CHANGED
```

```
      IF((KFUNC.NE.IFUNC).OR.(KARG.NE.IARG)) GO TO 50
C
C     SAVED INDEX CAN'T BE USED IF DIRECTION OF SEARCH HAS SWITCHED
      IF((IPOS*INDEX).LE.0) GO TO 50
      NDX=WRKSPC(INDXAD)
C
C     SAVED INDEX CAN'T BE USED IF VALUE AT SAVED INDEX HAS BEEN MOVED
      IF((NDX.AND.FL5MSK).NE.0)  GO TO 50
C
C     SAVED INDEX CAN'T BE USED IF VALUE AT SAVED INDEX HAS BEEN REMOVED
      IF((NDX.AND.FL1MSK).EQ.0)  GO TO 50
C
C     IS SEARCH FROM BEGINNING FASTER THAN FROM SAVED INDEX ?
      KNDEX=IABS(INDEX)
      IF(JPOS.LT.2) GO TO 50
      IF((JPOS+JPOS).LE.KNDEX)  GO TO 50
C
C     SAVED INDEX CAN BE USED, BEGIN SEARCH AT INDXAD.
      LOC=INDXAD
C     FIND RELATIVE DISTANCE FROM SAVED INDEX AND DETERMINE WHETHER TO
C     COUNT UP OR DOWN.  IF REQUESTED POSITION IS CLOSER TO THE BEGINNING
C     OF THE LIST THAN THE SAVED INDEX, COUNT UP, OTHERWISE, COUNT DOWN.
C
      LENGTH=INDEX-IPOS
      JPOS=IABS(LENGTH)
      IF(LENGTH) 10,28,40
C
COUNT UP FROM INDXADD
C
  40  ITOP=0
      GO TO 23
C
C     DO NOT USE SAVED INDEX.  START FROM THE BEGINNING OR END OF LIST
C
  50  WRKSPC(THIS)=WRKSPC(THIS).AND.NFLAG4
      IF(IPOS) 20,99,12
C
COUNT DOWN
C
  10  LOC=LVRTSH((WRKSPC(LOC).AND.MASK2),ISHFT2)
      IF((WRKSPC(LOC).AND.FL0MSK).NE.0)  GO TO 99
  12  IF(ITYP.EQ.3) GO TO 22
      ISTYP=(WRKSPC(LOC).AND.FLG67)
      IF(ISTYP.EQ.3)ISTYP=2
      IF(ISTYP.NE.ITYP) GO TO 10
  22  IND=IND+1
      IF(IND.NE.JPOS) GO TO 10
      GO TO 28
C
COUNT UP FROM THE BOTTOM OF THE LIST
C
  20  ITOP=1
  23  LOC=LVRTSH((WRKSPC(LOC).AND.MASK3),ISHFT3)
      IF(ITOP.EQ.1) GO TO 24
      KTEMP=LVRTSH((WRKSPC(LOC).AND.MASK2),ISHFT2)
      IF((WRKSPC(KTEMP).AND.FL0MSK).NE.0) GO TO 99
  24  ITOP=0
```

```
      IF(ITYP.EQ.3) GO TO 21
      ISTYP=WRKSPC(LOC).AND.FLG67
      IF(ISTYP.EQ.3)ISTYP=2
      IF(ISTYP.NE.ITYP) GO TO 23
   21 IND=IND+1
      IF(IND.NE.JPOS) GO TO 23
   28 IVAL=LVRTSH(WRKSPC(LOC),ISHFT1)
C
C     SAVE INDEX PARAMETERS AFTER SUCCESSFUL RETRIEVAL
C
   75 IF(NSKIP.EQ.1) GO TO 70
      KARG  =LVLFSH( IARG,ISHFT1)
      KFUNC =LVLFSH(IFUNC,ISHFT2)
      INDXAD=LVLFSH(LOC,ISHFT3)
      INDEX=IPOS.AND.MASK4
      INDEXS=KARG.OR.KFUNC.OR.INDXAD.OR.INDEX
      GO TO 70
C
C    FAILURE EXIT
   99 ITESTR=-1
      IF(NSKIP.NE.1) INDEXS=0
      IVAL=IARG
C
C    SUCCESS EXIT, SET DEFAULTS.
   70 ITYP=3
C    IS IVAL NEGATIVE?
      IVAL=SHIFT(IVAL, ISHFT1)
      IVAL=SHIFT(IVAL,-ISHFT1)
      RETURN
      END
```

```
      SUBROUTINE LVNSRT
      INTEGER WRKSPC,REGASP,FL0MSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,
     + FLG67,SEQSPC,THIS
     +,FLGTMP,TWO,THREE,HEAD,OLDLOC,ASPREG,TEMP,SVLRPL
      COMMON/LVVTR1/MEMSZE,REGASP,WRKSPC(1)
      COMMON /LVADDR/ IADD,THIS,LSTHED,LOC,LAST
      COMMON /LVARGS/IFUNC,IARG,IPOS,ITYP,IVAL,NVAL,NSKIP,ITESTR,INCLUD,
     + IVALS(10),ITYP1(10)
      COMMON/LVFLAG/FL0MSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,FLG67
      COMMON /LVVSEQ/ISEQSZ,ISQPOS,LASTSQ,SEQSPC(1)
      COMMON/LVMASK/MASK1,MASK2,MASK3,MASK4,NMASK1,NMASK2,NMASK3,NMASK4
      COMMON /LVSHFT/ ISHFT1,ISHFT2,ISHFT3
      DATA TWO/2B/,THREE/3B/,NFLG67/7777777777777777774B/,SVLRPL/0/
C
C     CALLS TO LVFIND OR LVFNV MUST PRECEDE A CALL TO THIS ROUTINE.
C
C     IS THE GIRS BUFFER FULL ?
 125  KTEMP=WRKSPC(REGASP).AND.MASK2
      KTEMP=LVRTSH(KTEMP,ISHFT2)
      IF(KTEMP.EQ.REGASP) GO TO 98
C
C    FORM FIRST WORD OF SINGLE OR MULTIVALUED FUNCTION
      FLGTMP=FL1MSK.OR.ITYP1(1)
      IF(NVAL.EQ.1)GO TO 20
      LSTTMP=REGASP
      FLGTMP=FLGTMP.OR.FL0MSK.OR.FL2MSK
      GO TO 21
   20 LSTTMP=IVALS(1).AND.777777B
C
C     IF THIS FUNCTION ALREADY EXISTS, GO TO 43
   21  IF(ITESTR.GT.0) GO TO 43
C
C    IF THAT ADDRESS IS ALREADY IN WORKING SPACE, GO TO 25
      IF((FL1MSK.AND.WRKSPC(IADD)).NE.0) GO TO 25
C
C     UPDATE REGASP(IF NECESSARY)
      KTEMP=MASK2.AND.WRKSPC(IADD)
      IF(IADD.EQ.REGASP)REGASP=LVRTSH(KTEMP,ISHFT2)
C
C    UPDATE AVAILABLE SPACE
      KTEMP1=WRKSPC(IADD).AND.MASK1
      KTEMP2=LVRTSH(KTEMP1,ISHFT1)
      WRKSPC(KTEMP2)=(WRKSPC(KTEMP2).AND.NMASK2).OR.KTEMP
      KTEMP=LVRTSH(KTEMP,ISHFT2)
      WRKSPC(KTEMP )=(WRKSPC(KTEMP ).AND.NMASK1).OR.KTEMP1
C
C     INSERT FUNCTION
      KTEMP=LVLFSH(IFUNC,ISHFT1)
      KTEMP1=LVLFSH(LSTTMP,ISHFT2)
      KTEMP2=LVLFSH(IADD,ISHFT3)
      WRKSPC(IADD)=KTEMP.OR.KTEMP1.OR.KTEMP2.OR.FL5MSK.OR.FLGTMP
C
C
C    INSERT ANY ADDITIONAL FUNCTION VALUES
      IF(NVAL.EQ.1) GO TO 100
      HEAD=IADD
      OLDLOC=IADD
```

```
      GO TO 50
C
C     IF THAT ADDRESS CONTAINS THE HEAD OF A CONFLICT LIST, GO TO 60
  25  IF((FL5MSK.AND.WRKSPC(IADD)).GT.0) GO TO 60
C
C   IF THAT ADDRESS CONTAINS A VALUE ON A MULTIVALUE LIST, GO TO 35
      IF((FL2MSK.AND.WRKSPC(IADD)).GT.0.AND.(FL0MSK.AND.WRKSPC(IADD)).EQ
     *.0) GO TO 35
C
C-------------------------------------------------------------------------
C-THE ADDRESS CONTAINS A FUNCTION ON A CONFLICT LIST,BUT NOT THE HEAD OF
      THIS=IADD
C
C   FIND THE PRECEDING FUNCTION ON THE CONFLICT LIST
C26   IF(LNKSPC(LNKSPC(THIS)).EQ.IADD)GO TO 27
  26  KTEMP=MASK3.AND.WRKSPC(THIS)
      KTEMP=LVRTSH(KTEMP,ISHFT3)
      KTEMP1=MASK3.AND.WRKSPC(KTEMP)
      KTEMP1=LVRTSH(KTEMP1,ISHFT3)
      IF(KTEMP1.EQ.IADD) GO TO 27
C     THIS=LNKSPC(THIS)
      KTEMP=WRKSPC(THIS).AND.MASK3
      THIS=LVRTSH(KTEMP,ISHFT3)
      GO TO 26
C27   LAST=LNKSPC(THIS)
  27  KTEMP=WRKSPC(THIS).AND.MASK3
      LAST=LVRTSH(KTEMP,ISHFT3)
      NEWLOC=REGASP
      KTEMP=WRKSPC(REGASP).AND.MASK2
      KTEMP=LVRTSH(KTEMP,ISHFT2)
      IF(KTEMP.EQ.REGASP) GO TO 98
C
C   UPDATE AVAILABLE SPACE AND REGASP
      CALL LVUPDT

C   MOVE THE FUNCTION ON A CONFLICT LIST TO THE FIRST CELL OF AVAILABLE
C     SPACE
      WRKSPC(NEWLOC)=WRKSPC(IADD).OR.FL4MSK
      WRKSPC(IADD)=0
C     LNKSPC(LAST)=NEWLOC
      KTEMP=LVLFSH(NEWLOC,ISHFT3)
      WRKSPC(LAST)=((WRKSPC(LAST).AND.NMASK3).OR.KTEMP)
C
C   INSERT THIS FUNCTION AS THE HEAD OF A CONFLICT LIST
      WRKSPC(IADD)=LVLFSH(IFUNC,ISHFT1)
      KTEMP1=LVLFSH(LSTTMP,ISHFT2)
      KTEMP=LVLFSH(IADD,ISHFT3)
      WRKSPC(IADD)=WRKSPC(IADD).OR.KTEMP.OR.KTEMP1.OR.FLGTMP.OR.FL5MSK
      IF((WRKSPC(NEWLOC).AND.FL0MSK).EQ.0)  GO TO 34
C
C   IF THE FUNCTION THAT WAS MOVED IS THE HEAD OF A MULTIVALUE LIST,FIX
C     NEXT=LSTSPC(NEWLOC)
      KTEMP=WRKSPC(NEWLOC).AND.MASK2
      NEXT=LVRTSH(KTEMP,ISHFT2)
C30   NEXT=LSTSPC(NEXT)
  30  KTEMP=WRKSPC(NEXT).AND.MASK2
      NEXT=LVRTSH(KTEMP,ISHFT2)
```

```
C      IF(LSTSPC(NEXT).NE.IADD)GO TO 30
C      LSTSPC(NEXT)=NEWLOC
       KTEMP=WRKSPC(NEXT).AND.MASK2
       KTEMP1=LVRTSH(KTEMP,ISHFT2)
       IF(KTEMP1.NE.IADD)GO TO 30
       KTEMP2=LVLFSH(NEWLOC,ISHFT2)
       WRKSPC(NEXT)=((WRKSPC(NEXT).AND.NMASK2).OR.KTEMP2)
C
C    INSERT ANY ADDITIONAL FUNCTION VALUES
  34   HEAD=IADD
       OLDLOC=IADD
       IF(NVAL.GT.1)GO TO 50
       GO TO 100
C
C-------------------------------------------------------------------------
C-THE ADDRESS CONTAINS A VALUE ON A MULTIVALUE LIST
  35   NEWLOC=REGASP
       KTEMP=WRKSPC(REGASP).AND.MASK2
       KTEMP=LVRTSH(KTEMP,ISHFT2)
       IF(KTEMP.EQ.REGASP) GO TO 98
C
C    UPDATE AVAILABLE SPACE AND REGASP
       CALL LVUPDT
C
C    MOVE THE VALUE ON A MULTIVALUE LIST TO THE FIRST CELL OF AVAILABLE S
       WRKSPC(NEWLOC)=WRKSPC(IADD)
       WRKSPC(IADD)=0
C
C      RESET POINTERS
C
C      L1=LSTSPC(NEWLOC)
       KTEMP=WRKSPC(NEWLOC).AND.MASK2
       L1=LVRTSH(KTEMP,ISHFT2)
       IF((FLOMSK.AND.WRKSPC(L1)).EQ.0) GO TO 200
C      LNKSPC(LSTSPC(L1))=NEWLOC
       KTEMP=WRKSPC(L1).AND.MASK2
       KTEMP1=LVRTSH(KTEMP,ISHFT2)
       KTEMP2=LVLFSH(NEWLOC,ISHFT3)
       WRKSPC(KTEMP1)=((WRKSPC(KTEMP1).AND.NMASK3).OR.KTEMP2)
       GO TO 201
C200   LNKSPC(L1)=NEWLOC
 200   KTEMP=LVLFSH(NEWLOC,ISHFT3)
       WRKSPC(L1)=((WRKSPC(L1).AND.NMASK3).OR.KTEMP)
C201   KZVAL=LSTSPC(LNKSPC(NEWLOC))
 201   KTEMP=WRKSPC(NEWLOC).AND.MASK3
       KTEMP1=LVRTSH(KTEMP,ISHFT3)
       KTEMP2=WRKSPC(KTEMP1).AND.MASK2
       KZVAL=LVRTSH(KTEMP2,ISHFT2)
       IF((WRKSPC(KZVAL).AND.FLOMSK).NE.0)  GO TO 38
C      LSTSPC(LNKSPC(NEWLOC))=NEWLOC
       KTEMP=LVLFSH(NEWLOC,ISHFT2)
       KTEMP1=WRKSPC(NEWLOC).AND.MASK3
       KTEMP2=LVRTSH(KTEMP1,ISHFT3)
       WRKSPC(KTEMP2)=((WRKSPC(KTEMP2).AND.NMASK2).OR.KTEMP)
       GO TO 39
C 38   LSTSPC(KZVAL)=NEWLOC
  38   KTEMP=LVLFSH(NEWLOC,ISHFT2)
```

```
      WRKSPC(KZVAL)=((WRKSPC(KZVAL).AND.NMASK2).OR.KTEMP)
  39  WRKSPC(IADD)=LVLFSH(IFUNC,ISHFT1)
C    INSERT THIS FUNCTION AS THE HEAD OF A CONFLICT LIST
C     LNKSPC(IADD)=IADD
C     LSTSPC(IADD)=LSTTMP
C     WRKSPC(IADD)=(FLGTMP.OR.WRKSPC(IADD))
C     WRKSPC(IADD)=(FL5MSK.OR.WRKSPC(IADD))
      KTEMP=LVLFSH(IADD,ISHFT3)
      KTEMP1=LVLFSH(LSTTMP,ISHFT2)
      WRKSPC(IADD)=WRKSPC(IADD).OR.KTEMP.OR.KTEMP1.OR.FLGTMP.OR.FL5MSK
      GO TO 100
C
C-------------------------------------------------------------------------
C-THE FUNCTION TO BE INSERTED IS ON THE CONFLICT LIST
  43  HEAD=THIS
C
C     IS THIS A SINGLE VALUE LIST OR MULTIVALUE LIST?
      IF(LSTHED.LT.0) GO TO 51
C
C     OLDLOC IS THE LOCATION OF THE LAST VALUE ON THE MULTIVALUE LIST
C
      KTEMP=WRKSPC(LSTHED).AND.MASK3
      OLDLOC=LVRTSH(KTEMP,ISHFT3)
C
C-------------------------------------------------------------------------
C-INSERT ADDITIONAL FUNCTION VALUES
 50   LSTASP=LVRTSH(WRKSPC(REGASP),ISHFT1)
      IN=0
      GO TO 56
C
C-------------------------------------------------------------------------
C-FORM MULTIVALUE LIST TO ADD VALUE(S) TO SINGLE-VALUED FUNCTION
  51  IN=0
      KTEMP=WRKSPC(REGASP).AND.MASK2
      KTEMP=LVRTSH(KTEMP,ISHFT2)
      IF(KTEMP.EQ.REGASP) GO TO 98
      LSTASP=LVRTSH(WRKSPC(REGASP),ISHFT1)
      NEWLOC=REGASP
C     REGASP=LSTSPC(REGASP)
      KTEMP=WRKSPC(REGASP).AND.MASK2
      REGASP=LVRTSH(KTEMP,ISHFT2)
C     NODSPC(NEWLOC)=LSTSPC(THIS)
      KTEMP=WRKSPC(THIS).AND.MASK2
      KTEMP1=LVLFSH(KTEMP,ISHFT1-ISHFT2)
      WRKSPC(NEWLOC)=((WRKSPC(NEWLOC).AND.NMASK1).OR.KTEMP1)
      TEMP=(WRKSPC(THIS).AND.FLG67)
      WRKSPC(NEWLOC)=(TEMP.OR.WRKSPC(NEWLOC))
C     WRKSPC(THIS)=(WRKSPC(THIS).AND.NFLG67)
C     WRKSPC(THIS)=(FL2MSK.OR.WRKSPC(THIS))
C     WRKSPC(THIS)=(FL0MSK.OR.WRKSPC(THIS))
      WRKSPC(THIS)=(((WRKSPC(THIS).AND.NFLG67).OR.FL0MSK).OR.FL2MSK)
      OLDLOC=THIS
C
C-------------------------------------------------------------------------
C   LVNSRT ANOTHER VALUE ON MULTIVALUE LIST
C52   WRKSPC(NEWLOC)=(FL2MSK.OR.WRKSPC(NEWLOC))
C     WRKSPC(NEWLOC)=(FL1MSK.OR.WRKSPC(NEWLOC))
```

```
  52   WRKSPC(NEWLOC)=((WRKSPC(NEWLOC).OR.FL1MSK).OR.FL2MSK)
C      LSTSPC(OLDLOC)=NEWLOC
       KTEMP=LVLFSH(NEWLOC,ISHFT2)
       WRKSPC(OLDLOC)=((WRKSPC(OLDLOC).AND.NMASK2).OR.KTEMP)
C      LNKSPC(NEWLOC)=OLDLOC
       KTEMP=LVLFSH(OLDLOC,ISHFT3)
       WRKSPC(NEWLOC)=((WRKSPC(NEWLOC).AND.NMASK3).OR.KTEMP)
       OLDLOC=NEWLOC
 56    NEWLOC=REGASP
       IF(IN.GT.0)GO TO 57
C
C    NO VALUES HAVE BEEN INSERTED YET
       IN=1
       GO TO 58
C
C    SOME VALUES HAVE BEEN INSERTED
 57    IF(IN.EQ.NVAL)GO TO 67
       IN=IN+1
C
    58 KTEMP=WRKSPC(REGASP).AND.MASK2
       KTEMP=LVRTSH(KTEMP,ISHFT2)
       IF(KTEMP.EQ.REGASP) GO TO 909
C 581 REGASP=LSTSPC(REGASP)
       KTEMP=WRKSPC(REGASP).AND.MASK2
       REGASP=LVRTSH(KTEMP,ISHFT2)
C 582 NODSPC(NEWLOC)=IVALS(IN)
       KTEMP=LVLFSH(IVALS(IN),ISHFT1)
       WRKSPC(NEWLOC)=((WRKSPC(NEWLOC).AND.NMASK1).OR.KTEMP)
       WRKSPC(NEWLOC)=(ITYP1(IN).OR.WRKSPC(NEWLOC))
       ITYP1(IN)=0
       GO TO 52
C
C    END MULTIVALUE LIST AND UPDATE AVAILABLE SPACE
C67    LSTSPC(OLDLOC)=HEAD
  67   KTEMP=LVLFSH(HEAD,ISHFT2)
       WRKSPC(OLDLOC)=((WRKSPC(OLDLOC).AND.NMASK2).OR.KTEMP)
C      NODSPC(REGASP)=LSTASP
       KTEMP=LVLFSH(LSTASP,ISHFT1)
       WRKSPC(REGASP)=((WRKSPC(REGASP).AND.NMASK1).OR.KTEMP)
C      LSTSPC(LSTASP)=REGASP
       KTEMP=LVLFSH(REGASP,ISHFT2)
       WRKSPC(LSTASP)=((WRKSPC(LSTASP).AND.NMASK2).OR.KTEMP)
C      LNKSPC(LSTSPC(HEAD))=OLDLOC
       KTEMP=WRKSPC(HEAD).AND.MASK2
       KTEMP1=LVRTSH(KTEMP,ISHFT2)
       KTEMP2=LVLFSH(OLDLOC,ISHFT3)
       WRKSPC(KTEMP1)=((WRKSPC(KTEMP1).AND.NMASK3).OR.KTEMP2)
       GO TO 100
C
C-------------------------------------------------------------------------
C-THE FUNCTION TO BE INSERTED IS NOT ON THE CONFLICT LIST
 60    ASPREG=REGASP
       LSTASP=LVRTSH(WRKSPC(REGASP),ISHFT1)
       KTEMP=WRKSPC(REGASP).AND.MASK2
       KTEMP=LVRTSH(KTEMP,ISHFT2)
       IF(KTEMP.EQ.REGASP) GO TO 98
C
```

```
C    UPDATE AVAILABLE SPACE AND REGASP
       CALL LVUPDT
C
C    INSERT FUNCTION IN FIRST CELL OF AVAILABLE SPACE
       KTEMP=LVLFSH(IFUNC,ISHFT1)
       WRKSPC(ASPREG)=((WRKSPC(ASPREG).AND.NMASK1).OR.KTEMP)
       IF(NVAL.EQ.1)GO TO 611
C      LSTSPC(ASPREG)=REGASP
       KTEMP=LVLFSH(REGASP,ISHFT2)
       WRKSPC(ASPREG)=((WRKSPC(ASPREG).AND.NMASK2).OR.KTEMP)
       WRKSPC(ASPREG)=(FL2MSK.OR.WRKSPC(ASPREG))
       WRKSPC(ASPREG)=(FL0MSK.OR.WRKSPC(ASPREG))
       GO TO 612
C611   LSTSPC(ASPREG)=IVALS(1)
 611   KTEMP=LVLFSH(IVALS(1),ISHFT2)
       WRKSPC(ASPREG)=((WRKSPC(ASPREG).AND.NMASK2).OR.KTEMP)
 612   WRKSPC(ASPREG)=ITYP1(1).OR.WRKSPC(ASPREG).OR.FL1MSK.OR.FL4MSK
       KTEMP=LVLFSH(IADD,ISHFT3)
       WRKSPC(ASPREG)=((WRKSPC(ASPREG).AND.NMASK3).OR.KTEMP)
       KTEMP=LVLFSH(ASPREG,ISHFT3)
       WRKSPC(LAST)=((WRKSPC(LAST).AND.NMASK3).OR.KTEMP)
       IF(NVAL.EQ.1) GO TO 100
C
C    INSERT ADDITIONAL VALUES
       LSTASP=LVRTSH(WRKSPC(REGASP),ISHFT1)
       OLDLOC=ASPREG
       HEAD=ASPREG
       IN=0
       GO TO 56
C
C      DESTRUCTIVE INSERTION
C
       ENTRY LVDSIN
C
C     A CALL TO LVFIND MUST PRECEDE A CALL TO EITHER ENTRY POINT.
C     GIVEN N VALUES OF TYPE K ON A LIST WHERE N.GE.0 , INDEXED
C     INSERTIONS SHALL SUCCEED FOR IPOS.GE.1 .AND. IPOS.LE.N+1
C
C     DEFEAT SAVED INDEX UNTIL NEXT RETRIEVAL.
       WRKSPC(THIS)=WRKSPC(THIS).OR.FL4MSK
       JPOS=IABS(IPOS)
       KPOS=IPOS
       INDEX=0
C     DOES THE IPOS'TH VALUE OF THE PROPER TYPE EXIST?
       IF(ITESTR.LT.0) GO TO 90
C     REPLACE VALUE AT LOCATION 'LOC'.  SVL OR MVL?
       IF(LSTHED.GT.0) GO TO 356
C     SVL
       WRKSPC(LOC)=(WRKSPC(LOC).AND.NMASK2).OR.LVLFSH(IVALS(1),ISHFT2)
       SVLRPL=1
       GO TO 365
C     MVL
 356   WRKSPC(LOC)=(WRKSPC(LOC).AND.NMASK1).OR.LVLFSH(IVALS(1),ISHFT1)
C     REPLACE TYPE.
 365   WRKSPC(LOC)=((WRKSPC(LOC).AND.NFLG67).OR.ITYP1(1))
       GO TO 100
C
```

```
C     IPOS'TH VALUE WAS NOT FOUND, INDEXED INSERTION CAN STILL SUCCEED
C     IF (IPOS-1) VALUE IS FOUND.  THIS THEN BECOMES A NORMAL INSERTION
C     IF JPOS=1 OR THE VALUE WILL BE THE LAST IN THE LIST.  OTHERWISE,
C     THIS BECOMES A NONDESTRUCTIVE INSERTION TO THE FIRST POSITION IN
C     THE LIST
C
  90   IF(JPOS.EQ.1) GO TO 125
       IF(KPOS) 91,97,92
  91   KPOS=KPOS+1
       GO TO 93
  92   KPOS=KPOS-1
  93   CALL LVFIND
       IPOS=KPOS
       CALL LVFNV(INDEX)
C     FAILURE IF NO VALUE IS FOUND.
       IF(ITESTR.LT.0) GO TO 97
C     NORMAL INSERTION IF REQUEST WAS IPOS'TH FROM THE TOP.
       IF(KPOS.GT.0) GO TO 125
C     NONDESTRUCTIVE INSERTION AT THE BEGINNING OF THE LIST.
       NEWLOC=REGASP
       CALL LVUPDT
C     SVL OR MVL?
       IF(LSTHED.GT.0) GO TO 377
       GO TO 344
C
C      NONDESTRUCTIVE INSERTION
C
       ENTRY LVNDIN
C
C     IF IPOS=-1, PLACE AT THE END OF THE LIST (NORMAL INSERTION).
       IF(IPOS.EQ.-1) GO TO 125
C
C     DEFEAT SAVED INDEX UNTIL NEXT RETRIEVAL.
       WRKSPC(THIS)=WRKSPC(THIS).OR.FL4MSK
       JPOS=IABS(IPOS)
       KPOS=IPOS
       INDEX=0
       NEWLOC=REGASP
C     DOES THE IPOS'TH VALUE OF THE PROPER TYPE EXIST?
       IF(ITESTR.LT.0) GO TO 90
       CALL LVUPDT
C     SVL OR MVL?
       IF(LSTHED.LT.0) GO TO 344
C     MVL
       IF(KPOS.LT.0) GO TO 347
C
C     PLACE VALUE AT THE IPOS'TH POSITION (WRT ITYP) FROM THE TOP OF LIST
C377   ISTLOC=LNKSPC(IADD)
 377   KTEMP=WRKSPC( LOC).AND.MASK3
       ISTLOC=LVRTSH(KTEMP,ISHFT3)
C      NODSPC(NEWLOC)=IVALS(1)
C      LNKSPC(NEWLOC)=ISTLOC
       WRKSPC(NEWLOC)=LVLFSH(IVALS(1),ISHFT1)
       KTEMP1=LVLFSH( LOC,ISHFT2)
       WRKSPC(NEWLOC)=(((WRKSPC(NEWLOC).OR.KTEMP).OR.KTEMP1).OR.
      + (FLGTMP.OR.FL2MSK))
       IF(LOC.NE.LSTHED) GO TO 321
```

```
C      LSTSPC(LSTSPC(ISTLOC))=NEWLOC
       KTEMP=LVLFSH(NEWLOC,ISHFT2)
       KTEMP1=LVRTSH((WRKSPC(ISTLOC).AND.MASK2),ISHFT2)
       WRKSPC(KTEMP1)=((WRKSPC(KTEMP1).AND.NMASK2).OR.KTEMP)
       GO TO 322
C 321  LSTSPC(ISTLOC)=NEWLOC
  321  KTEMP=LVLFSH(NEWLOC,ISHFT2)
       WRKSPC(ISTLOC)=((WRKSPC(ISTLOC).AND.NMASK2).OR.KTEMP)
  322  KTEMP=LVLFSH(NEWLOC,ISHFT3)
       WRKSPC( LOC)=((WRKSPC( LOC).AND.NMASK3).OR.KTEMP)
       GO TO 100
C
C     PLACE VALUE AT THE IPOS'TH POSITION (WRT ITYP) FROM THE BOTTOM OF
C     THE LIST
C 347 NODSPC(NEWLOC)=IVALS(1)
C      LNKSPC(NEWLOC)=IADD
  347  WRKSPC(NEWLOC)=LVLFSH(IVALS(1),ISHFT1)
       KTEMP=WRKSPC( LOC).AND.MASK2
       KTEMP1=LVLFSH( LOC,ISHFT3)
       WRKSPC(NEWLOC)=WRKSPC(NEWLOC).OR.KTEMP.OR.KTEMP1.OR.FLGTMP.OR.
      + FL2MSK
C      IF((WRKSPC(LSTSPC(IADD)).AND.FL0MSK).EQ.0) GO TO 323
       KTEMP2=LVRTSH(KTEMP,ISHFT2)
       IF((WRKSPC(KTEMP2).AND.FL0MSK).EQ.0) GO TO 323
C      KZVAL=MASK2.AND.WRKSPC( LOC)
C      LNKSPC(LSTSPC(KZVAL))=NEWLOC
       KTEMP3=LVLFSH(NEWLOC,ISHFT3)
       KTEMP4=WRKSPC(KTEMP2).AND.MASK2
       KTEMP5=LVRTSH(KTEMP4,ISHFT2)
       WRKSPC(KTEMP5)=((WRKSPC(KTEMP5).AND.NMASK3).OR.KTEMP3)
       GO TO 324
C 323  LNKSPC(LSTSPC(IADD))=NEWLOC
  323  KTEMP=LVLFSH(NEWLOC,ISHFT3)
       KTEMP1=WRKSPC( LOC).AND.MASK2
       KTEMP2=LVRTSH(KTEMP1,ISHFT2)
       WRKSPC(KTEMP2)=((WRKSPC(KTEMP2).AND.NMASK3).OR.KTEMP)
C 324  LSTSPC(IADD)=NEWLOC
  324  KTEMP=LVLFSH(NEWLOC,ISHFT2)
       WRKSPC( LOC)=((WRKSPC( LOC).AND.NMASK2).OR.KTEMP)
       GO TO 100
C
C     CREATE MVL WITH NEW VALUE AT THE TOP OF THE LIST.
  344  KTEMP=LVRTSH((WRKSPC(REGASP).AND.MASK2),ISHFT2)
       IF(REGASP.EQ.KTEMP) GO TO 99
       NWLOC2=REGASP
       CALL LVUPDT
C      NODSPC(NEWLOC)=IVALS(1)
C      LSTSPC(NEWLOC)=NWLOC2
C      LNKSPC(NEWLOC)=NWLOC2
       WRKSPC(NEWLOC)=LVLFSH(IVALS(1),ISHFT1)
       KTEMP=LVLFSH(NWLOC2,ISHFT2)
       KTEMP1=LVLFSH(NWLOC2,ISHFT3)
       WRKSPC(NEWLOC)=WRKSPC(NEWLOC).OR.KTEMP.OR.KTEMP1.OR.FL1MSK.OR.
      + FL2MSK.OR.ITYP1(1)
       KTEMP=MASK2.AND.WRKSPC(THIS)
       KTEMP1=LVLFSH(KTEMP,ISHFT1-ISHFT2)
       KTEMP2=LVLFSH(THIS,ISHFT2)
```

```
C      LNKSPC(NWLOC2)=NEWLOC
       KTEMP3=LVLFSH(NEWLOC,ISHFT3)
       KLGTEP=WRKSPC(THIS).AND.FLG67
       WRKSPC(NWLOC2)=(KTEMP1.OR.KTEMP2).OR.(KTEMP3.OR.KLGTEP).OR.
     + (FL1MSK.OR.FL2MSK)
C      LSTSPC(IADD)=NEWLOC
       KTEMP=LVLFSH(NEWLOC,ISHFT2)
       WRKSPC(THIS)=WRKSPC(THIS).AND.NMASK2
       WRKSPC(THIS)=(WRKSPC(THIS).OR.FL0MSK).OR.(FL2MSK.OR.KTEMP)
       GO TO 100
  98   ITESTR=-4
       PRINT 20001
20001  FORMAT(  * ERROR...THERE IS NO ADDITIONAL SPACE FOR THE GRAPH, THE
     * PROGRAM IS TERMINATED*)
       STOP
  99   ITESTR=-3
       PRINT 2, NVAL
 2     FORMAT(6H ONLY ,I4,28H VALUE(S) HAVE BEEN INSERTED)
 22    FORMAT(1X,I5,1H(,I5,35H) USED LAST CELL OF AVAILABLE SPACE)
       GO TO 97
 909   PRINT 22,IFUNC,IARG
C     THIS INSERTION HAS FILLED GIRS MEMORY - CALL A USER SUPPLIED
C     PROGRAM - LVEXIT.
       ITESTR=-2
       GO TO 97
 100   KTEMP=LVRTSH((WRKSPC(REGASP).AND.MASK2),ISHFT2)
       IF(REGASP.EQ.KTEMP) GO TO 909
C     FLAG 4 IS SET BECAUSE THIS INSERTION MIGHT BE A RECREATION OF AN
C     OLD LIST
       WRKSPC(THIS)=WRKSPC(THIS).OR.FL4MSK
       IVAL=IVALS(1)
C     "FAILURE" RETURN IF FUNCTION DID NOT PREVIOUSLY EXIST
       IF(((FLGSPC(THIS).AND.FL0MSK).NE.0).OR.SVLRPL.EQ.1) ITESTR=1
  97   IPOS=1
       ITYP=3
       NVAL=1
       SVLRPL=0
       ITYP1(1)=0
       RETURN
       END



       SUBROUTINE LVUPDT
       INTEGER WRKSPC,REGASP
       COMMON/LVVTR1/MEMSZE,REGASP,WRKSPC(1)
       COMMON/LVMASK/MASK1,MASK2,MASK3,MASK4,NMASK1,NMASK2,NMASK3,NMASK4
       COMMON /LVSHFT/ ISHFT1,ISHFT2,ISHFT3
C
C     THIS ROUTINE UPDATES AVAILABLE SPACE AND THE REGISTER OF AVAILABLE
C     SPACE - REGASP
C
C      LSTSPC(NODSPC(REGASP))=LSTSPC(REGASP)
C      NODSPC(LSTSPC(REGASP))=NODSPC(REGASP)
C      REGASP=LSTSPC(REGASP)
       KTEMP=WRKSPC(REGASP).AND.MASK1
       KTEMP1=LVRTSH(KTEMP,ISHFT1)
       KTEMP2=WRKSPC(REGASP).AND.MASK2
       KTEMP3=LVRTSH(KTEMP2,ISHFT2)
       WRKSPC(KTEMP1)=(WRKSPC(KTEMP1).AND.NMASK2).OR.KTEMP2
       WRKSPC(KTEMP3)=(WRKSPC(KTEMP3).AND.NMASK1).OR.KTEMP
       REGASP=KTEMP3
       RETURN
       END
```

```
      SUBROUTINE LVDLET
      INTEGER WRKSPC,REGASP,FL0MSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,
     + FLG67,SEQSPC,THIS
      COMMON/LVVTR1/MEMSZE,REGASP,WRKSPC(1)
      COMMON /LVADDR/ IADD,THIS,LSTHED,LOC,LAST
      COMMON /LVARGS/IFUNC,IARG,IPOS,ITYP,IVAL,NVAL,NSKIP,ITESTR,INCLUD,
     + IVALS(10),ITYP1(10)
      COMMON/LVFLAG/FL0MSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,FLG67
      COMMON /LVVSEQ/ISEQSZ,ISQPOS,LASTSQ,SEQSPC(1)
      COMMON/LVMASK/MASK1,MASK2,MASK3,MASK4,NMASK1,NMASK2,NMASK3,NMASK4
      COMMON /LVSHFT/ ISHFT1,ISHFT2,ISHFT3
      DATA NFLG02/7777777777777777537B/
C
C     DELETE ENTIRE LIST.  CALL FIND TO DETERMINE SVL OR MVL, LOCATION
C     OF FUNCTION AND FIRST VALUE.  FAILURE RETURN IF NO LIST.
C
      CALL LVFIND
      IF(ITESTR.LT.0) RETURN
      IF(LSTHED.LT.0) GO TO 2
C     DELETE ENTIRE MULTIVALUE LIST
      ISADD=LSTHED
      LOC=THIS
    5 NXTADD=WRKSPC(ISADD).AND.MASK2
      NXTADD=LVRTSH(NXTADD,ISHFT2)
      WRKSPC(ISADD)=0
      WRKSPC(ISADD)=WRKSPC(REGASP).AND.MASK1
      KTEMP=LVLFSH(REGASP,ISHFT2)
      WRKSPC(ISADD)=WRKSPC(ISADD).OR.KTEMP
      KTEMP=LVRTSH(WRKSPC(REGASP),ISHFT1)
      KTEMP1=LVLFSH(ISADD,ISHFT2)
      WRKSPC(KTEMP)=(WRKSPC(KTEMP).AND.NMASK2).OR.KTEMP1
      KTEMP=LVLFSH(ISADD,ISHFT1)
      WRKSPC(REGASP)=(WRKSPC(REGASP).AND.NMASK1).OR.KTEMP
      IF((WRKSPC(NXTADD).AND.FL0MSK).NE.0) GO TO 2
      ISADD=NXTADD
      GO TO 5
C
C     DELETE SINGLE VALUED FUNCTION
C     IS THE FUNCTION HEAD OF A CONFLICT LIST
    2 IF(THIS.NE.IADD) GO TO 68
      NXFUNC=WRKSPC(IADD).AND.MASK3
      NXFUNC=LVRTSH(NXFUNC,ISHFT3)
C     IF THIS FUNCTION IS THE ONLY ONE ON THE CONFLICT LIST, GO TO 10.
C     OTHERWISE, PLACE NEXT FUNCTION ON CONFLICT LIST IN 'HEAD OF
C     CONFLICT LIST' LOCATION (IADD)
      IF(NXFUNC.EQ.IADD) GO TO 10
      WRKSPC(IADD)=WRKSPC(NXFUNC).OR.FL5MSK
      IF((WRKSPC(IADD).AND.FL0MSK).EQ.0) GO TO 9
C     IF THE MOVED FUNCTION IS A MVL, THE POINTER FROM THE LAST VALUE OF
C     THE LIST TO THE HEAD MUST BE UPDATED.
      KVAL=MASK2.AND.WRKSPC(IADD)
      KVAL=LVRTSH(KVAL,ISHFT2)
    8 KVAL=MASK2.AND.WRKSPC(KVAL)
      KVAL=LVRTSH(KVAL,ISHFT2)
      KTEMP=WRKSPC(KVAL).AND.MASK2
      KTEMP=LVRTSH(KTEMP,ISHFT2)
      IF((WRKSPC(KTEMP).AND.FL0MSK).EQ.0) GO TO 8
```

```
      KTEMP=LVLFSH(IADD,ISHFT2)
      WRKSPC(KVAL)=(WRKSPC(KVAL).AND.NMASK2).OR.KTEMP
    9 LOC=NXFUNC
C     RETURN LOCATION TO AVAILABLE SPACE
   10 WRKSPC( LOC)=WRKSPC(REGASP).AND.MASK1
      KTEMP=LVLFSH(REGASP,ISHFT2)
      WRKSPC( LOC)=(WRKSPC( LOC).AND.NMASK2).OR.KTEMP
      KTEMP=LVLFSH( LOC,ISHFT1)
      WRKSPC(REGASP)=(WRKSPC(REGASP).AND.NMASK1).OR.KTEMP
      KTEMP=LVRTSH(WRKSPC( LOC),ISHFT1)
      KTEMP1=LVLFSH( LOC,ISHFT2)
      WRKSPC(KTEMP)=(WRKSPC(KTEMP).AND.NMASK2).OR.KTEMP1
      RETURN
C
C     FUNCTION TO BE DELETED IS NOT THE HEAD OF A CONFLICT LIST.
C     THE FUNCTION PRECEDING THIS (FUNCTION BEING DELETED) MUST POINT TO
C     THE FUNCTION FOLLOWING THIS
   68 KTEMP=WRKSPC(THIS).AND.MASK2
      WRKSPC(LAST)=(WRKSPC(LAST).AND.NMASK2).OR.KTEMP
      GO TO 10
C
      ENTRY LVDLTI
C
C     THIS ENTRY POINT WILL HANDLE INDEXED DELETION.
C
C     FUNCTION MUST BE A MVL OR, IF SVL, ABS(IPOS)=1 WITH PROPER TYPE.
C     OUTPUT IS EXPECTED FROM LVFIND.
C     DOES THE FUNCTION EXIST ?
      IF(ITESTR.LT.0) RETURN
C     SVL OR MVL ?
      IF(LSTHED.LT.0) GO TO 2
C
C     DELETE VALUE AT LOC.  DEFEAT SAVED INDEX FOR THIS LIST UNTIL AFTER
C     NEXT RETRIEVAL.
      WRKSPC(THIS)=WRKSPC(THIS).OR.FL4MSK
C
C     INDEXED DELETE CAN BE REDUCED TO FOUR CASES.  DELETE VALUE IN
C     FIRST, MIDDLE, OR LAST POSITION ON LIST, OR REDUCE TO SVL.
C
      NEXT  =LVRTSH((WRKSPC(  LOC).AND.MASK2),ISHFT2)
      NPRIOR=LVRTSH((WRKSPC(   LOC).AND.MASK3),ISHFT3)
C
C     IS LOC THE  LAST POSITION IN THE LIST ?
      IF(NEXT.EQ.THIS) GO TO 80
C
C     IS LOC THE FIRST POSITION IN THE LIST ?
      KTEMP =LVRTSH((WRKSPC(NPRIOR).AND.MASK2),ISHFT2)
      IF(KTEMP.EQ.THIS) GO TO 70
C
C     VALUE IS IN A MIDDLE POSITION IN THE LIST.  RECONNECT VALUES
C     PRECEEING AND FOLLOWING LOC.
C
      WRKSPC(NPRIOR)=(WRKSPC(NPRIOR).AND.NMASK2).OR.LVLFSH(NEXT,ISHFT2)
      WRKSPC(NEXT)=(WRKSPC(NEXT).AND.NMASK3).OR.LVLFSH(NPRIOR,ISHFT3)
      GO TO 10
C
C     DELETE VALUE IN LAST POSITION IN LIST
```

```
   80  WRKSPC(NPRIOR)=(WRKSPC(NPRIOR).AND.NMASK2).OR.LVLFSH(NEXT,ISHFT2)
       NEXT1 =LVRTSH((WRKSPC(  NEXT).AND.MASK2),ISHFT2)
       WRKSPC(NEXT1)=(WRKSPC(NEXT1).AND.NMASK3).OR.LVLFSH(NPRIOR,ISHFT3)
       GO TO 60
C
C      DELETE VALUE IN FIRST POSITION IN LIST
   70  WRKSPC(NEXT)=(WRKSPC(NEXT).AND.NMASK3).OR.LVLFSH(NPRIOR,ISHFT3)
       WRKSPC(THIS )=(WRKSPC(THIS ).AND.NMASK2).OR.LVLFSH(NEXT  ,ISHFT2)
C
C      CONVERT TO A SINGLE VALUE LIST ?
C
   60  KTEMP=LVRTSH((WRKSPC(NPRIOR).AND.MASK3),ISHFT3)
       IF(KTEMP.NE.NPRIOR) GO TO 10
C      IF DELETING LAST VALUE, RESET NEXT TO FIRST VALUE
       IF(NEXT.EQ.THIS) NEXT=NPRIOR
       KTEMP=WRKSPC( NEXT).AND.MASK1
       KTEMP=LVRTSH(KTEMP,ISHFT1-ISHFT2)
       WRKSPC(THIS)=(WRKSPC(THIS).AND.NMASK2).OR.KTEMP
       WRKSPC(THIS)=(WRKSPC(THIS).OR.(WRKSPC( NEXT).AND.MASK4)).AND.
      + NFLG02
       WRKSPC(NEXT )=WRKSPC(REGASP).AND.MASK1
       KTEMP=LVLFSH(REGASP,ISHFT2)
       WRKSPC( NEXT)=(WRKSPC(NEXT ).AND.NMASK2).OR.KTEMP
       KTEMP=WRKSPC( NEXT).AND.MASK2
       KTEMP =LVRTSH(KTEMP,ISHFT2)
       KTEMP1=LVLFSH( NEXT,ISHFT1)
       WRKSPC(KTEMP)=(WRKSPC(KTEMP).AND.NMASK1).OR.KTEMP1
       KTEMP=WRKSPC( NEXT).AND.MASK1
       KTEMP =LVRTSH(KTEMP,ISHFT1)
       KTEMP1=LVLFSH( NEXT,ISHFT2)
       WRKSPC(KTEMP)=(WRKSPC(KTEMP).AND.NMASK2).OR.KTEMP1
       GO TO 10
       END
```

```
      SUBROUTINE LVDUMP(KK,JJ,L)
      INTEGER WRKSPC,WORKSP,BINFIL,SEQSPC,REGASP
      COMMON/LVVTR1/MEMSZE,REGASP,WRKSPC(1)
      COMMON/LVRAND/ KPRIME,KSEED,NROW,KDNODE,KDROW,KTEMP
      COMMON/LVVTR5/BINFIL,KOMPAN,WORKSP(1)
      COMMON /LVTABL/ MAPSZE,MAP(1)
      COMMON/LVFLAG/FL0MSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,FLG67
      COMMON /LVVSEQ/ISEQSZ,ISQPOS,LASTSQ,SEQSPC(1)
      COMMON/LVMASK/MASK1,MASK2,MASK3,MASK4,NMASK1,NMASK2,NMASK3,NMASK4
      COMMON /LVSHFT/ ISHFT1,ISHFT2,ISHFT3
      IF(JJ.EQ.0) GO TO 50
      K=KK
      J=JJ
      IF(KK.LT.1) K=1
      IF(JJ.GT.MEMSZE) J=MEMSZE
      WRITE(L,10)
   10 FORMAT(1H1,*      GIRS MEMORY DUMP (IN OCTAL)*,///)
      WRITE(L,20) REGASP,MEMSZE,KPRIME,KSEED,NROW,KDNODE,KTEMP,KDROW,
     + ISEQSZ,MAPSZE
   20 FORMAT(1X,* REGASP=*,I6,/* MEMSZE=*,I6,6X,*PRIME=*,I3,6X,*SEED=*
     +,I3,6X,*NROW=*,I6,6X,6X,*KDNODE=*,I6,6X,*TEMP=*,I6,6X,
     +*KDROW=*,I6,6X,/,1X,*SEQSIZE=*,I6,6X,*MAPSIZE=*,I6,///)
      WRITE(L,30)
   30 FORMAT(1X,*                NODSPC                LSTSPC
     *     LNKSPC          FLGSPC  OCTAL COUNTER*,///)
      DO 40 I=K,J
      N1=LVRTSH(WRKSPC(I),ISHFT1)
      N2=WRKSPC(I).AND.MASK2
      N2=LVRTSH(N2,ISHFT2)
      N3=WRKSPC(I).AND.MASK3
      N3=LVRTSH(N3,ISHFT3)
      N4=WRKSPC(I).AND.MASK4
      WRITE(L,15)I,N1,N2,N3,N4,I
   40 CONTINUE
   15 FORMAT(1X,I6,2X,O20,2X,O20,2X,O20,2X,O6,2X,O8)
      RETURN
   50 WRITE(L) REGASP,MEMSZE,KPRIME,KSEED,NROW,KDNODE,KTEMP,KDROW,
     + ISEQSZ,MAPSZE
      WRITE(L)(WRKSPC(M),M=1,MEMSZE)
      WRITE(L)(SEQSPC(I),I=1,ISEQSZ)
      RETURN
      END
```

```
      SUBROUTINE LVPACK(NODSPC,LSTSPC,LNKSPC,L)
      INTEGER WRKSPC,REGASP,FL0MSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,
     + FLG67,SEQSPC,THIS
      COMMON/LVVTR1/MEMSZE,REGASP,WRKSPC(1)
      COMMON/LVRAND/ KPRIME,KSEED,NROW,KDNODE,KDROW,KTEMP
      COMMON/LVFLAG/FL0MSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,FLG67
      COMMON /LVVSEQ/ISEQSZ,ISQPOS,LASTSQ,SEQSPC(1)
      COMMON/LVMASK/MASK1,MASK2,MASK3,MASK4,NMASK1,NMASK2,NMASK3,NMASK4
      COMMON /LVSHFT/ ISHFT1,ISHFT2,ISHFT3
      DIMENSION NODSPC(1),LSTSPC(1),LNKSPC(1)
C
C     THIS ROUTINE PACKS A GIRS BUFFER WHICH WAS CREATED WITH
C     THE UNPACKED VERSION.
C
      READ(L) REGASP,MEMSZE,KPRIME,KSEED,NROW,KDNODE,KTEMP,KDROW,
     + ISEQSZ,MAPSZE
      READ(L)(NODSPC(I),I=1,MEMSZE)
      READ(L)(LSTSPC(I),I=1,MEMSZE)
      READ(L)(LNKSPC(I),I=1,MEMSZE)
      READ(L)(WRKSPC(I),I=1,MEMSZE)
      READ(L)(SEQSPC(I),I=1,ISEQSZ)
      DO 10 I=1,MEMSZE
C
C    IF THIS IS A HOLLERITH VALUE, MASK OUT BLANKS.
      IF((WRKSPC(I).AND.2).EQ.0) GO TO 11
      IF((WRKSPC(I).AND.FL2MSK).EQ.0) GO TO 13
      NODSPC(I)=NODSPC(I).AND.MASK1
      GO TO 14
   13 NODSPC(I)=LVLFSH(NODSPC(I),ISHFT1)
      LSTSPC(I)=LVRTSH((LSTSPC(I).AND.MASK1),(ISHFT1-ISHFT2))
      GO TO 12
   11 NODSPC(I)=LVLFSH(NODSPC(I),ISHFT1)
   14 LSTSPC(I)=LVLFSH(LSTSPC(I),ISHFT2)
   12 LNKSPC(I)=LVLFSH(LNKSPC(I),ISHFT3)
      WRKSPC(I)=((WRKSPC(I).OR.NODSPC(I)).OR.(LSTSPC(I).OR.LNKSPC(I)))
   10 CONTINUE
      RETURN
      END
```

```
      FUNCTION LVRTSH(IWRD,IBITS)
      DATA MASK/377777777777777777778/
***         WRITTEN BY M. CHERNICK     CODE 832
      LVRTSH=SHIFT(IWRD,-IBITS)
      IF(IWRD.GE.0)GO TO 10
      MSK=SHIFT(MASK,-IBITS+1)
      LVRTSH=LVRTSH.AND.MSK
   10 RETURN
      END
```

```
      FUNCTION LVLFSH(IWRD,IBITS)
***THIS ROUTINE PERFORMS A LEFT LOGICAL SHIFT
***         WRITTEN BY M. CHERNICK     CODE 832
      DATA MASK/4000000000000000000B/
      IF(IBITS.EQ. 0)  GO TO 10
      IF(IBITS.EQ.60)  GO TO 12
      LWRD= SHIFT(IWRD,IBITS)
      IF(IBITS.EQ.59)  GO TO 15
      MSK= SHIFT(MASK,IBITS-59)
      LVLFSH=LWRD.AND.MSK
      RETURN
   15 LVLFSH=LWRD.AND.MASK
      RETURN
   12 LVLFSH=0
      RETURN
   10 LVLFSH=IWRD
      RETURN
      END
```

The following BLOCK DATA statement is
need only for the PDP 11 implementation

```
      BLOCK DATA
      IMPLICIT INTEGER(A-Z)
      COMMON/LVFLAG/FL0MSK,FL1MSK,FL2MSK,FL3MSK,FL4MSK,FL5MSK,FLG67
      DATA FL0MSK/'200/,FL1MSK/'100/,FL2MSK/'40/,FL5MSK/'4/,FLG67/'3/,
     1 FL3MSK/'20/,FL4MSK/'10/
      END
```

REFERENCES

1. Berkowitz, S., "Design Trade-offs for a Software Associative Memory," DTNSRDC Report 3531 (May 1973).

2. Berkowitz, S., "Graph Information Retrieval Language; Programming Manual for FORTRAN Complement; Revision One," DTNSRDC Report 76-0085 (Feb 1976).

3. Berkowitz, S., "PIRL - Pattern Information Retrieval Language - Design of Syntax," Proceedings of 1971 National Conference of the Association for Computing Machinery (1971) pp. 496-507.

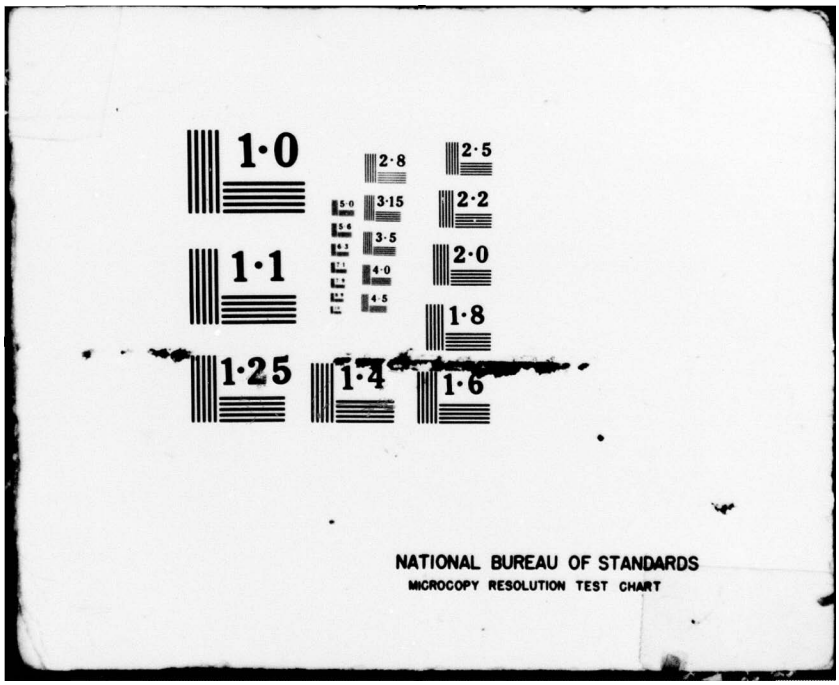
PRECEDING PAGE BLANK-NOT FILMED

AD-A068 204 DAVID W TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CE--ETC F/G 9/2
GIRS (GRAPH INFORMATION RETRIEVAL SYSTEM) USERS MANUAL.(U)
APR 79 I S ZARITSKY

UNCLASSIFIED DTNSRDC-79/036 NL

3 OF 3
ADA
068204

END
DATE
FILMED
6-79
DDC


1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·5
2·8
3·15
3·5
4·0
4·5
5·0
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

INITIAL DISTRIBUTION

Copies	
1	CHONR/430D, M. Denicoff
1	NRL
1	NSWC
1	NUSC
1	NOSC
1	NAVSUP/0414, G. Bernstein
3	NAVSEC 1 SEC 6114 1 SEC 6178D
1	Rome Air Development Center
12	DDC

CENTER DISTRIBUTION

Copies	Code	Name
1	18	G. Gleissner
1	1802.2	F. Frenkiel
1	1803	S. Rainey
1	1804	L. Avrunin
1	1805	E. Cuthill
1	1806	J. Pulos
2	1809	D. Harris
1	182	A. Camara
1	1821	D. Jefferson
1	1822	T. Rhodes
1	1824	S. Berkowitz
1	1824	J. Carlberg
1	1824	J. Garner
1	1824	P. Marques
1	1824	W. Parsons
35	1824	I. Zaritsky
1	1826	L. Culpepper
1	1828	C. Godfrey
1	1828	W. Gorham, Jr.
1	1828	M. Wallace
1	1828.1	I. Datz
1	184	H. Lugt
1	184.1	H. Feingold
1	1843	J. Schot
1	1844	S. Dhir
1	1844	J. McKee
1	185	T. Corin
1	1850	A. Cinque
1	1851	J. Brainin
1	1854	H. Sheridan
1	1855	R. Brengs
1	187	R. Ploe
1	187	M. Zubkoff
1	189	G. Gray
1	189.1	N. Taylor
10	5211.1	Reports Distribution
1	522.1	
1	522.2	

PRECEDING PAGE BLANK-NOT FILMED